A Stylus-Based User Interface for Text: Entry and Editing

by

Aaron Goodisman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1991
[June 1991]

Signature of Author .....
Department of Electrical Engineering and Computer Science
10 May 1991

Certified by .....
David Goldberg
Research Scientist, Xerox Palo Alto Research Center
Thesis Supervisor

Certified by .....
Christopher Schmandt
Principal Research Scientist, Media Laboratory
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman of the Committee On Graduate Students

A Stylus-Based User Interface for Text: Entry and Editing

by

Aaron Goodisman

Submitted to the Department of Electrical Engineering and Computer Science
on 10 May 1991, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Science

Abstract

A computer system with a user interface based on a stylus offers many potential benefits. A stylus is portable, usable with one hand, and works with a wide variety of systems, from notebook-sized computers to computers with wall-sized displays.

In any new system, the methods by which a user manipulates textual information are important. This thesis explores the utility of stylus-based input for several text-related tasks, and informally studies a number of user interaction techniques.

We describe a system for entering text with a stylus and investigate user interface techniques for interfacing with a text recognizer, concluding that a stylus is a feasible input device for entering small amounts of text. We also implement a simple text editing system utilizing gestural commands and explore the interactions of a stylus with some additional user interface techniques: scrolling and on-screen buttons. We discuss some alternatives in the design of such an editing system, including the use of "markup editing."

We conclude that stylus-based systems can be easy to use and learn and lend themselves to the incorporation of knowledge about users' tasks.

Thesis Supervisor: David Goldberg
Title: Research Scientist, Xerox Palo Alto Research Center

Thesis Supervisor: Christopher Schmandt
Title: Principal Research Scientist, Media Laboratory

Acknowledgements

I gratefully acknowledge the contributions of David Goldberg, my supervisor at PARC, to this thesis. Some of the ideas described here are his, and many others we developed jointly. In addition, Mr. Goldberg provided many valuable references.

He and Russel Brown, a summer intern at PARC in 1990, developed the hand printing recognition system used in the text entry described here.

I would also like to thank both of my supervisors, Chris Schmandt and David Goldberg, for their extensive comments on this document, as well as all the people at PARC who discussed these ideas with me, agreed to be test users, and generally supported my stay there.

Contents

List of Figures

Foreword

In August of 1989, the Computer Science Laboratory (CSL) at Xerox Corporation's Palo Alto Research Center (PARC) decided to change its charter to focus on "ubiquitous computing." Their vision is that the future holds a world in which objects with computational power are everywhere, communicating amongst themselves and with people. No one will go to a computer; there will always be a computationally active surface nearby, already linked to any necessary computational resources.

In such a world a keyboard and mouse will often be an inappropriate interface to the computational network. This thesis, on the use of a stylus as a device for interacting with computers, is a small step in the transition to the world of ubiquitous computing.

Chapter 1

Introduction

A stylus-sensitive surface mounted over the display of a computer system allows users to "write" directly on the screen. With the advances made in miniaturization and display technology, a wide variety of new systems can be created, from notepad-sized computers to computers with screens that cover entire walls. The stylus has the potential to be the one input device that can work gracefully across this entire range of systems, allowing them to mimic the notepads and whiteboards on which they are based.

Initial investigation, however, reveals that such a stylus-based interface is not trivial to develop. This thesis explores some of the issues arising in the development of a stylus-based interface, in particular for the entry and editing of textual information.

1.1 Outline of this thesis

This chapter begins by giving some motivations for the development of a user interface based solely (or primarily) on a stylus. It then describes the study undertaken towards this end, including methodology.

Chapter 2 explores some of the work that has already been done in this and

related fields.

Chapter 3 focuses on the entry of text into a computer system using a stylus and some of the issues that arise in this topic.

This leads to chapter 4 on the editing of text with a stylus.

Chapter 5 describes some experiments in "markup editing," in which the user's actions do not actually modify the text but only mark the document until the user is satisfied with the proposed changes.

Finally, chapter 6 contains the conclusions we drew in this work.

1.2 Motivation for a stylus-based system

There are many motivations for developing a system using solely stylus-based input. For instance, a stylus is suited to a wider range of computer uses and user interface tasks than most other input devices, both conventional and less common. Stylus-based systems can also be fast and easy to learn and use.

1.2.1 Applicable in many scenarios

Computers can be built in many ways. A computer can be a stationary, desktop machine; with the current advances in miniaturization, it can also be a portable, full-function, notebook-sized computer. Or it can have a large, wall-sized screen and be used as an electronic version of a whiteboard. In the future, a computer could also be as small as a checkbook or watch.

There are a variety of situations in which a computer can be used. Computers can be used as single person workstations for office work, or as organizers for personal information. A computer with a wall-sized display can be used by one person or by several people together. Small, portable computers can be used while traveling, in meetings, or in other situations where a desktop or wall-mounted machine may not be available or appropriate.

A keyboard and mouse are well suited to a desktop machine, as they require a smooth, flat, stable, horizontal surface. And they are well suited for use as a single person workstation. They are not so well suited to other scenarios, however. They are difficult to carry. And it is hard to imagine using a keyboard and mouse with a computer whose screen is the size of a checkbook, a watch, or a wall. This last example is especially interesting. A keyboard and mouse could be used with a wall-sized system from twenty feet away if the screen were simply a blowup of a conventional screen, but more effective use of such screen space would be for many people to work on it simultaneously, moving in front of it, sharing information and resources. A keyboard and mouse do not fit in well with this image. Keyboard and mouse systems are also very obtrusive and are sometimes inappropriate, in meetings for example.

Voice input, on the other hand, could work with a system of any size, since it does not put limitations on the physical characteristics of the system. It makes just as much sense to use voice with a notebook computer as it does to use voice with a wall-sized computer. But, like keyboard input, voice input is not always appropriate. In a business meeting, for example, there are many reasons not to use voice, among them that it might be distracting or that the spoken information might be confidential.

A stylus does not suffer from these difficulties. Like voice, it works well across the range of systems, but a stylus does not broadcast its information. It makes sense to use a stylus with a notebook computer, with a desktop computer, and with a computer with a large display. It even makes sense for several people to use multiple styluses simultaneously on the same computer.

Because a stylus works across the range of devices, it provides a consistent interface as a user moves between different sy ms. And because a stylus is small, a user can carry one around or one can be mounted easily near the display of each system. In addition, the stylus requires only one hand to operate, leaving

the user's other hand free for carrying a portable computer or for performing other activities.

1.2.2 Applicable to many tasks

There are several different sorts of tasks for which the range of computer systems described above can be used. One task is entering a large amount of text; sometimes it is necessary to get a lot of information into a computer. At other times, only a small amount of text needs to be entered; e.g. specifying a file name or text string for which to search. A third task is indicating a position in a document or on the screen, or selecting a region of a document for extracting, printing, or as part of an editing process. And, of course, a system must be able to receive commands and instructions.

A keyboard is good for entering text, both in large and small quantities, if its user is a good typist. If not, the process can be slow and tedious. A keyboard is also fairly good for entering commands, although users sometimes have difficulty remembering the many command names and their semantics. By itself, though, a keyboard is not a good device for positioning and selection tasks, because a user must either indicate positions by reference (line numbers, etc.) or by tediously directing the movements of a cursor.

The addition of a mouse allows positions and selections to be indicated much more easily, but introduces the problem that a user must move one hand back and forth between the mouse and the keyboard. A mouse can also be used to press on-screen buttons, to make selections from menus, or to invoke iconically represented programs, greatly simplifying the specification of commands.

Voice input, on the other hand, has the potential to be a much faster text entry technique than typing on a keyboard, although the necessary technology is not yet perfected.[1] It is poorly suited, however, to other tasks. It is difficult to indicate a

[1]People can speak comfortably at between 150 and 200 words per minute, as compared with 40

two dimensional position on a screen using voice. And a command that is simple to invoke with another input technology can be prohibitively awkward when spoken; e.g. "Move cursor to end of line."

Other devices also have their problems. A thumb wheel, for example, can replace the functionality of a mouse, and without the necessity for a nearby flat surface. Its use is limited, however, without a keyboard, since there is no good way to enter text with a thumb wheel; it is even harder to write with a thumb wheel than with a mouse.

A stylus is a good compromise. It is not well suited for large amounts of text entry, but a small amount is certainly feasible: a user can write the necessary text by hand. And a stylus is even better for positioning and selection tasks than a mouse or thumb wheel because a user indicates positions by placing the stylus directly on the screen at the desired location. There is no cursor acting as an intermediary.

Plus, a stylus has the potential to be a better device for specifying commands than a mouse, since it can be used not only to press buttons or to select from menus, but also to draw symbolic marks representing commands and, in some cases, their parameters.

Touch sensitive surfaces are much like stylus sensitive surfaces, except that they do not require a user to have specialized hardware. Although it might be possible to use a touch system like a stylus system to specify commands, such a system would be poor for other tasks, such as text entry and positioning, because a touch system lacks the sharp point and fine dynamic control of a stylus-based system.

---

to 60 words per minute typing. These statistics are cited as "self-measurement" in an unpublished manuscript by Mark Stefik, *Towards the Digital Workspace* ([30]).

1.2.3 Ease of use and learning

In addition to being broadly applicable, a stylus-based system is easy to learn and use. A stylus-based system is easy to learn because it involves no unfamiliar skills. Gaining familiarity with a keyboard-based system requires a significant investment of time for people who do not type. Other unconventional interaction devices also require training. A "chord" keyboard might take even longer to learn than a normal keyboard. And getting used to a thumb wheel is no trivial task. A stylus-based system takes advantage of skills its users already have: using a pen or pencil.

A stylus-based system can make use of writing and drawing. Such a system can be simple and self-explanatory. And since it is an easy system to learn, it can also be an easy system to use. Because a stylus is well suited for direct positioning tasks, many options can be presented to a user at once, without fear of confusion or difficulty in expressing a choice among them. With a thumb wheel or mouse, it is much easier to position the cursor accidentally over an incorrect selection.

1.2.4 Speed

If well designed, a stylus-based system can be very efficient to use. A user can communicate a great deal to the system with a single drawn symbol, making use of its position, orientation, shape, and even the style in which it is drawn. With one horizontal line, for example, an extraneous phrase can be removed from a document. Rather than typing out lengthy command names, or trying to remember cryptic, short ones, users can take advantage of natural, symbolic commands, remembering them not only with visual memory, but also with the memory of the muscles.

1.2.5 Overall comparison to other devices

Like many input technologies, a stylus is well suited to a broad range of systems and situations. It is superior to other devices, though, particularly a keyboard and mouse, by being portable, usable with one hand, and appropriate in many situations where other technologies are not. In addition, a stylus-based system is well suited to the tasks for which a computer is used, with the exception of the entry of large amounts of text. Indeed, a stylus-based system is better for these tasks than most devices: any amount of text entry with a thumb wheel is difficult and voice is poorly suited for positioning.

Because a stylus is more widely applicable and better for most tasks than many other devices, it is a good choice as the primary input device for new systems that cannot easily be used with a keyboard and mouse. In addition, a stylus-based system can be easy and fast to learn and use by taking advantage of users' existing skills and knowledge.

1.3 This study

Computer systems are used for many things, from graphics arts to cartography to VLSI design, but an important part of most applications is text processing. A tremendous volume of information is available on computers, and much of it is in textual form. Users search this text, read sections of it, change it, send it to other users, and create their own new text. In addition, text is used to specify file names and to label drawings, diagrams, and maps.

While there are also documents of other types (graphical, audio/video, mixed, etc.), textual documents are such an overwhelming majority that any new interaction device must ensure its usability with this medium. Consequently, the focus of this thesis is on stylus-based interactions specifically with text.

We investigate the use of a stylus for entering and editing text. The goal was

to develop interaction techniques for text manipulation with a stylus, but not to compare the stylus with other input media.

For our experiments we had a stylus-sensitive surface mounted over a Liquid Crystal Display (LCD) and attached as a peripheral to a Sun workstation. Thus software mockups could be created that used the stylus on the peripheral display screen as if it were a self-contained system, but with the computational power and development environment of the workstation.

1.3.1 Scope

Stylus-based interactions with text is a fairly broad area. Text comes in a variety of styles (bold, italics, etc.) as well as fonts and sizes. And a stylus could be a very good device for handling this complexity. We felt, however, that we should limit this initial study to plain text in a single size and font. Once this topic is reasonably well understood, it will make a good foundation for studies of documents with multiple fonts, sizes, and styles.

1.3.2 Methodology

The methods used for this investigation were not those of detailed user studies. We felt that a scientific study would require more effort than would be justified by the precise results obtained. Instead, we took the approach of implementing software that used the stylus on the display screen and then observing various people using it, both in unstructured sessions and while performing a particular task. With the insights gained from these observations, we improved the software systems. This cycle was repeated several times.

Chapter 2

Previous and Related Work

A significant amount of work has been done that is directly or indirectly related to this study. In addition to the large body of literature on the automatic interpretation of hand drawn text, there has been a great deal of investigation of gestural interfaces. A variety of other studies about input technology are also relevant.

2.1 Entering text by hand

For more than twenty years people have been attempting to solve the problems associated with entering text into a computer system by drawing it with a pen or electronic equivalent. Although there are many works on this topic ([35], [4], [14], etc.), it is not a central focus of our study, and we will not discuss this literature at any length. The reader is referred instead to the survey works that appear periodically, such as those by Suen, Berthod and Mori ([31]) and Tappert, Suen, and Wakahara ([34]).

2.2 Specifying commands with a stylus

A significant amount has been written on topics related to giving instructions to a computer system by means of hand drawn marks. We have divided this literature into several groups, but most works contain something related to each category.

2.2.1 People and drawing

Some of the literature focuses primarily on people's ability to use hand drawn commands. Wolf ([38]) explores the ease of such commands and the consistency people have with themselves and each other when using them. With Morrel-Samuels, she also studies the use of gestures especially for text editing ([39]) in some experiments with paper and pencil.

Gould and Salaun discuss more general work on hand markings ([18]) and describe a number of experiments (also using paper and pencil) in the manipulation of both text and graphics. And Morrel-Samuels discusses the distinction between lexical and gestural commands ([25]).

2.2.2 Examples

Many people have built actual systems to experiment with these ideas. Some were built over twenty years ago, such as those by Sutherland ([32]) and Coleman ([11]), while others are more recent. Buxton, Fiume, Hill, Lee, and Woo ([5]) discuss gesture driven input in the context of an editor for sketches. Welbourne and Whitrow ([36]) describe a gesture based text editor, while Kurtenbach and Buxton ([23]) describe an editor for graphics, especially interesting for its use of circling as a selection method. Another text editor and some of the issues involved in its creation and use is described by Kankaanpaa ([20]). There are many similar systems, but by far the most complete one is the Penpoint operating system, developed by the GO Corporation ([9], [8]). This system has been developed

specifically for computers whose sole input device is a pen. It includes hand print recognition and gestural commands.

2.2.3 Interpreting marks

Some work focuses more directly on the task of interpreting the marks made by users. Rhyne ([26]), for example, discusses the management of dialogues in gestural interfaces. In particular, however, we found Dean Rubine's PhD thesis from Carnegie Mellon University ([27]) to be most useful. This work is extremely complete and we refer back to it later.

2.2.4 Other related work

There have also been a variety of useful studies of different input technologies. Goodwin ([16]), for example, compares the lightpen (a type of stylus) to the light-gun and keyboard in three basic tasks. And Gould and Alfaro ([17]) simulate editing text with several different input techniques and compare the results. Karat, McDonald, and Anderson ([21]) compare several ways to select items from a menu, and Jackson and Roske-Hofstrand ([19]) explore circling as a method for selecting objects in a document. Brown, Buxton, and Murtagh ([3]) experiment with some novel ways to use an off-screen tablet and Whitfield, Ball, and Bird ([37]) compare off-screen and on-screen touch devices.

Finally, there was additional work that seemed useful, such as a discussion of *The Reactive Keyboard*, a typing prediction system by Darragh, Witten, and James ([12]), discussed in section 3.6.3. We were also interested in work on multiple representations of documents, such as those by Chen and Harrison ([10]), and Brooks ([2]), discussed in section 5.3.2.

Overall the literature was helpful, but there seemed to be insufficient study of the user interface issues related to the entry and editing of text.

Chapter 3

Text Entry

Text entry with a stylus is usually considered to consist mostly of hand printing or handwriting recognition, interpreting the marks a user makes with the stylus as text. There is much more to it than that, however.

One unresolved issue is when to interpret the text the user writes: Never? When the user is finished entering the text? Or as the text is being entered? We consider the last to make the most sense and implement a system of this sort to explore ways to make text entry natural for users.

The hand print recognition systems used in this sort of interpretation need a way to divide the users' input into single characters. Some characters consist of only one stroke; others have several. How can the system determine which strokes belong to which characters? Writing text into rows of boxes divides the strokes into characters, but is unsatisfying. Rather than using an algorithmic way to solve this problem, we explore a user interface solution.

Another issue is how to handle mistakes in the interpretation of the entered text. How can the frequency of these mistakes be reduced? How can users easily correct them? We explore the use of heuristics to reduce misinterpretation and propose a general correction paradigm.

Some systems for recognizing hand entered text are trained to users' writing.

This training can be tedious and does not always reflect the actual ways users enter text. We explore a method for training the system during the text entry process.

More generally, there are many issues involved with making text entry seem natural, allowing it to follow the user's normal process of composition. Users should not be limited in how much text they can enter at once and should be able to edit the text they are entering. We describe some features of our system along these lines.

3.1 When to recognize

When text is being entered by hand, there are several possible times that the system could interpret the text and convert it to a form more easily manipulable by a computer. One choice is for the system never to interpret the entered text. That is, the text could remain in the form of a drawing, either a bitmap pattern or a series of consecutive points as traced by the tip of the stylus. Alternately, the system could convert the drawing into text after a user has finished writing. That is, the user writes a complete message or section, signifies to the system that the text is complete, and then the system converts the hand drawn marks into text. A third choice is for the system to interpret the text as a user is entering it. While the user makes marks with the stylus, the previous marks are interpreted and displayed as text on the screen.

Unrecognized text has only a few advantages over paper. A short note sent from one person to another could be in this form, but could be delivered electronically instead of physically. Another use might be a scrawled reminder set to pop up on a user's screen at a particular time; unlike paper, this note can draw attention to itself.

These uses have problems, though. Inevitably, the messages get saved or refer-

enced and a user desires to search through them or to manipulate their contents in some textual way. Or part of a message originally intended as transient becomes relevant on a longer time scale and a user wishes to copy the text into some other document. Since the text is uninterpreted, though, it is much more difficult for the computer to do any sort of matching or formatting.

In addition, the interpreted version of a block of hand-entered text is much more compact than the uninterpreted version. If past history is representative, more and more information will be stored on computers and keeping this information in its ASCII representation uses significantly less storage than keeping it in a hand entered form.

Interpreted text is not only more useful and compact than uninterpreted text; it is also more flexible. Interpreted text can be displayed in any font or size, and is easily reformatted for display in various system configurations. The question then is: When should the text be interpreted?

The Penpoint system interprets text when the user signifies that the text entry is complete, for example by pressing the "ok" button in a text entry window. Although this allows for flexibility in entering the text, there are problems with leaving the text uninterpreted until its entry is complete.

One such problem is handling errors in the interpretation of the text. Automatic hand drawn text interpretation is, by nature, imperfect, because hand printing and handwriting contain inherent ambiguities. For example, a hand printed symbol might lie somewhere between two letters, such as an 'n' and an 'h'. Humans are able to read hand drawn text by making use of higher level information about the expected content of the text, such as spelling and grammar. Although computer systems can make use of some of this information (discussed below in section 3.3.2), they are far from the sophistication of human readers. Consequently, the text produced by the system when the user has finished writing may not be precisely the text that the user intended, and some method for correction must be included.

Correction or editing of plain text with a stylus is not difficult to imagine, except that it must include a way to enter text, at the very least to add letters that the system dropped, or replace letters that the system recognized incorrectly. But what if the text entered as a correction needs correcting? Furthermore, the established paradigm for text entry in this sort of system is geared towards entering of large blocks of text, and is consequently awkward for entering a single character or word, such as might be necessary in a correction. Should a new text entry window pop up on the screen when one character needs replacing? Of course, different text entry methods could be used for bulk entry and for correction. Perhaps bulk text could be entered by hand and corrections chosen from a menu or palette.

This approach has problems. The entry of text requires the user to go through two distinct phases: the initial entry of the text and the correction of that text. Furthermore, the method of text entry in those two phases is different: in the initial phase text entry is done by hand drawing the text, while in the correction phase text entry is done by some other method.

An additional difficulty with not interpreting the text until a user has finished writing it is that this does not allow the user to edit the text being entered. It is one thing to correct a few misrecognized characters; it is another thing to add a sentence or change several words. Because it is very difficult to edit the text before it has been interpreted, such editing would probably be left to the correction phase. But this is awkward, as it is desirable to be able to change the entered text while it is being entered; while the user's thoughts are on each partially completed sentence.

Our conclusion, then, is that text should be interpreted as the user is entering it. This text can easily be edited and new text can be inserted smoothly, since the established entry method is for the system to interpret the text automatically shortly after the user enters it.

3.2 Segmenting

Of course, the recognition techniques used for the different interpretation paradigms could be different. A recognizer for large blocks of hand drawn text could attempt to recognize large pieces of the text at once, perhaps with some method of word or phrase shape recognition, whereas an "on-the-fly" recognizer is more likely to attempt to recognize individual characters.

In the recognition of individual characters, it is necessary to segment the drawn strokes; that is, to determine which strokes constitute a single character. The easiest way to do this is to require that each character be drawn in a separate box, such as in filling out a form. Figure 3-1 shows a line of boxes, in which several characters have been entered, some of which have been interpreted. These boxes and the feeling that entering text is like filling out a form, however, are not desirable in many applications.

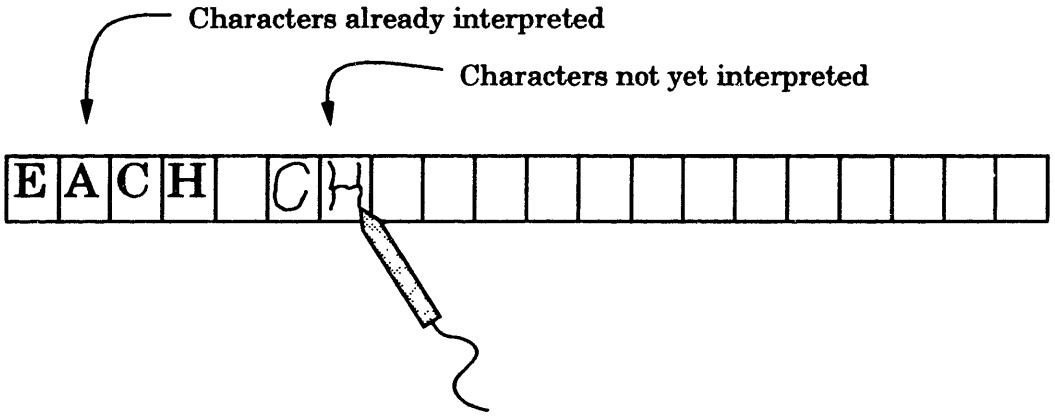

Figure 3-1: Drawing each character in a separate box.

An additional problem is that the system has no way of knowing how much text a user wishes to enter. The system might present one or more rows of boxes, into which text can be written, such as in figure 3-2. If the user intends to enter only a small amount of text, such as a single word or character, then there are many

extraneous boxes. If the user intends to write several paragraphs, then there are most likely too few boxes. In either case, the user is entering only one character at a time, so almost none of the boxes need to be visible.

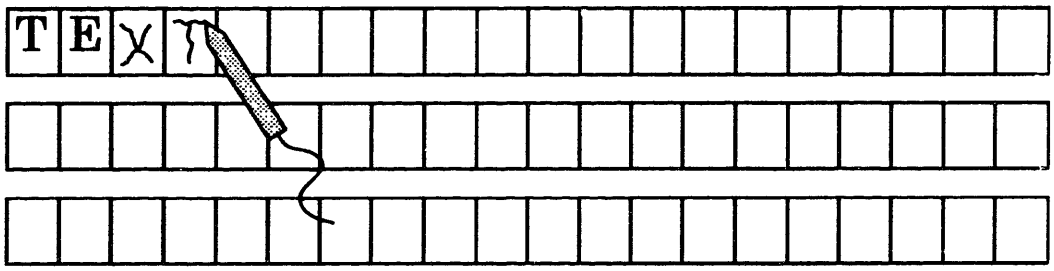

Figure 3-2: Several rows of boxes, into which text can be written.

3.2.1 The segmenting cursor

Instead, only the box into which a user is writing a character needs to be visible. After the user has entered that character, the box moves to the next position and the user can enter the next character.[1] In our system, text is entered into boxes one character at time and the boxes move as the characters are entered.

The problem with this idea as we have described it so far is one of synchronization. How does the system know when to move the box to the next position? Must the user wait for the system to realize a character is completed, or is the user required to keep up as the box moves ahead at its own rate? Neither of these techniques is desirable.

A slightly different version is one in which both the box in which a user is writing and the boxes into which the user is likely to write next are visible, as in figure 3-3. Thus when the user has completed a character, there is no delay before moving on to the next character. When the user begins writing in a new

[1]This idea thanks to David Gifford of the MIT Laboratory for Computer Science.

box, the system can safely assume that the user has finished with the character in the previous box, and move the boxes. The system provides not only the box immediately following the current character, but additional boxes in case the user wants to skip a space or move to the next line.

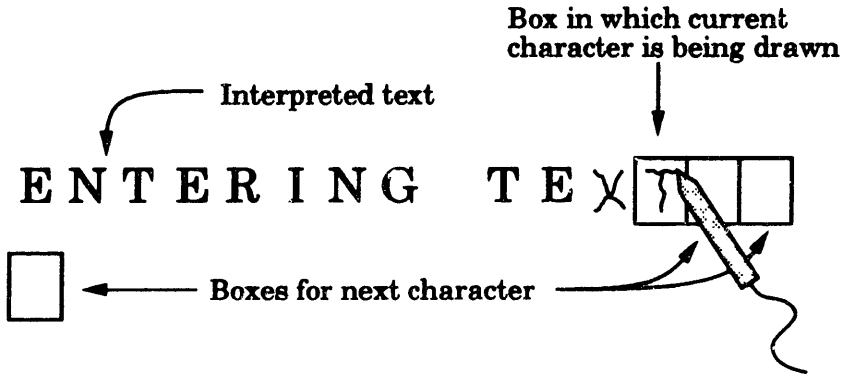

Figure 3-3: Displaying only the box in which a user is writing and the boxes into which the user is likely to write next.

These boxes behave much like a traditional cursor, "running" along the leading edge of the entered text, marking the insertion point. They have the additional feature, however, of segmenting the user's marks into distinct characters, easy for the system to interpret. This is the derivation of the name "segmenting cursor."

In addition, the segmenting cursor tells the system when to interpret a character. If a user puts the pen down outside of the current box, then the character in that box is most likely completed and can be interpreted.[2]

Another complexity of the segmenting cursor arose, however, when users accidentally put the pen down in the wrong box. A user might intend a stroke to lie in one box, but accidentally begin it in an adjacent box. As described so far,

[2]Admittedly, this is not always true. Some people finish a word before dotting their i's or crossing their t's. Although we did not have the opportunity to experiment with this, the system could delay the interpretation of text until several characters have been entered or until the user has skipped a space, if that were more effective. We did not encounter any problems with interpreting text immediately, though, once we explained the constraints to our users.

the segmenting cursor moves (and any uninterpreted character is analyzed) as soon as the pen touches down. To compensate for this problem, we modified the system so that a character is not interpreted until one stroke of the next character drawn has been completed or one second has passed. In this way, if a user begins a stroke in one box, but the stroke lies primarily in another, the system moves the segmenting cursor to the location the user most likely intends and does not yet interpret the character at that location (in case there are more strokes to draw for that character). If the user's hand moves away from the screen, however, any uninterpreted character is converted to text after one second has passed.

The segmenting cursor prevents text entry from seeming like form completion, but is still an effective way to separate characters. Our test users had no trouble using it to enter both text we specified and text of their own devising.

3.3 Misrecognition

Hand drawn text is ambiguous not only to computers, but also to humans, including sometimes even the person who originally wrote it. It is no wonder, then, that this type of software occasionally misrecognizes characters. Our system includes a simple heuristic, based on the expected input, to reduce the frequency of misrecognition and a general paradigm for correcting mistakes that is simple, effective, and applicable to the correction of a wide variety of errors.

3.3.1 Our recognition software

The recognition software available for this study has several models of each character.[3] It interprets a group of strokes by comparing them to every model and choosing the best match. It requires, however, that the group of strokes comprise exactly one character.

[3]Please see Acknowledgments, page 3.

It is efficient enough that it can run on a Sun SPARCStation-1 and leave more than enough computational power for tracking the stylus and handling other routine activity while still performing its recognition in real-time. That is, it is capable of recognizing characters as fast as we can draw them.

In addition, it can return not just the character that best matches a drawn symbol, but the best several matches, with an indication of the degree of matching. This will become important in the discussion below of reducing and correcting misrecognition.

3.3.2 Reducing misrecognition

Obviously, it is desirable to reduce the number of characters recognized incorrectly by the system. Although we did not pursue this line of study extensively, we attempted to determine if simple heuristics based on knowledge of the expected input could make the system significantly more accurate. If the text to be entered were known to consist only of digits, for example, the system could be much more accurate, since there would be fewer characters to distinguish.

Rather than attempting to make a comprehensive partitioning of possible input sets, however, we decided to ascertain merely whether the accuracy would improve if the input were known to be English text.

For this purpose we use a table of English letter digraph frequencies generated from a large quantity of available text.[4] The statistics thus obtained are of the form $P_{xy} = P_y|P_x$; that is, for any two characters, x and y, the probability of character y appearing in the text given that the previous character is known to be character x.

When interpreting a character, then, we used a heuristic based on these probabilities. Rather than generating the single best interpretation for a drawn character, the hand printing recognition software generates the four best interpretations

[4]An on-line version of Grolier's Encyclopedia and the New York Times monthly summaries for most of 1990 – about 220 megabytes of text.

with an indication of their relative fits with the user's marks. When several of these possible interpretations are roughly equally good matches with the drawn character, and one is more than 100 times more likely to follow the previous character than the others, then the likely character is chosen as the interpretation.

For example, if the user were to write something that could be either an 'n' or an 'h', when the previous character is a 'c', then the system would interpret the marks as an 'h', even if the 'n' were a slightly better match, because the probability of an 'h' following a 'c' is over 1400 times higher than the probability of an 'n' following a 'c'.[5] This does not mean that it is difficult for a user to write "cn" because the heuristic only fires if there is reasonable doubt as to what symbol the user drew.

This heuristic is fairly effective. The system seems much more accurate, although there are still occasional errors (especially confusion between similar letters that could function similarly in a word, such as 'a' and 'o', which are similar in some people's printing). Naturally, the system is much more accurate when entering English text than when entering non-English strings of letters and numbers.

Our experiments with this heuristic showed two things. One is that this particular heuristic is useful for significantly reducing misrecognition at a very low computational cost. The other is that such heuristics can be added on top of a hand printing or handwriting recognition system to provide a benefit without being integrated into the recognition system itself. That is, different heuristics can be used for different classes of expected input but with the same recognition software. The improvement in accuracy can be layered on top of a basic system.

3.3.3 Correcting misrecognition

Even with some knowledge of the expected input, a text entry system will still sometimes come up with an interpretation of a user's writing other than what the

[5]According to the statistics we generated, the probability of an 'n' following a 'c' is 0.000090, while the probability of an 'h' following a 'c' is 0.133017.

user intended. This may be caused by ambiguities in the system, or it may occur when a user writes a particular character messily. Whatever the reason, there are times when the interpreted text contains errors.[6] Even the best systems have accuracy rates of around 95%.[7]

In order for these errors to be corrected, they must first be identified. Although it would be desirable for the system to identify errors, it is very difficult, since an error is simply a place where the interpreted text and the user's intended text do not match. For example, the system could check the spelling of each word in the interpreted text. This would provide some hints as to which portions of the text were correct and which were not, but it would not be at all reliable. The word "cat" is spelled correctly, but might still be an error if the user intended "cot," and "Media-lab" or "PARC" might not be in a spelling list, but might not be errors either.

For reasons of this sort, it must therefore be left to the user to identify errors. We now determine how the user should signify these errors to the system and what the system should do once an error has been identified.

Whatever a user does to point out an error should be easy and quick. When writing text with a stylus, the user should be able to correct misrecognitions while writing, without having to stop, correct, and then try to get back on the track of the text. We propose that the user simply tap the tip of the stylus once on any errors to mark them. This is a quick motion and one that is easy to do accurately.

When the user indicates an erroneously recognized character, the system should attempt to correct it. If the recognition software's first choice for the character is not correct, one of its other choices often is. And if two of these other choices are equally good matches with the drawn character, the same heuristics used to pick the first choice can be used to choose between them.

---

[6] We exclude those errors which occur when a user writes a character other than the one intended.

[7] Appendix C of "The Power of Penpoint" ([8]) lists the character recognition of the Penpoint system as 94%, for example.

Consequently, as a user writes, occasional characters are interpreted incorrectly, but the user periodically taps on the incorrect characters and they almost always become correct.

We experimented with other ways of making use of the different possible interpretations of each character but found the above technique to be the most effective. Displaying the system's best guess for an alternate interpretation was too distracting; either the alternate characters were large enough to pull the user's attention away from the task at hand or they were too small to be useful. We tried putting each alternate letter in a small button near the original character and allowing the user to press the button to select an alternate letter, but found this awkward as well.

The system's alternate choice for a character might be wrong, so tapping on the original error might not have fixed it. In this case we found that it made more sense to have the user redraw the character than to present any more alternates. In a well-trained system any character not recognized as either the system's first or second choice is probably not drawn well enough for the system ever to recognize it, and the user will do much better to redraw it.[8]

Thus, if a user taps on an already corrected letter, the system moves the segmenting cursor (the box in which characters are drawn) back to the location of the mis-entered character and allows the user to re-enter it. Then the cursor returns to its previous position.

In practice we found that users redrew characters more frequently than we had expected. This was partly because they occasionally had trouble using the hardware and thus drew badly formed characters (especially when they were first introduced to the system), and partly because this was an easy way to correct

[8]This is not always true. In many people's handwritings, the uppercase 'O', lowercase 'o', and zero are all identical, so the system's second choice might not be correct, even though the character was well drawn. The heuristics based on expected input prevent this from causing problems in most cases, however.

errors that *they* had made (as distinct from errors the system had made), such as drawing an 'a' when they intended an 'o'.

We had not originally intended a double tap to mean "replace character," but it fit in well with the other actions we included for correction and editing of the entered text. (See section 3.5 for more on editing text while entering it.)

Obviously there are other things the system could do to correct errors once a user identifies them. The system could look for words spelled similarly to the erroneous word in a dictionary. Then it could either display them for the user to select or make an intelligent guess as to which word the user intended (perhaps making use of the alternate choices for characters in the incorrect word, grammatical analysis of the text, etc.). Although we did not experiment further along these lines, correction with the tapping gesture is easily expandable to a variety of "intelligent" or user-guided correction mechanisms. Tapping does not specify the scope of an error, nor does it specify what method the system should use to correct the error. Consequently, it can be used to signify any error, and an intelligent system can determine both the scope and an appropriate correction method for the error. Thus a user must remember only one gesture for indicating errors and can be confident that the system will respond reasonably, either by correcting the error or by providing choices of corrections or correction methods.

3.4 Tuning model letters with user interactions

The hand print recognition system we used works by comparing the marks drawn by the user with a number of models, and determining how well the user's marks match each model. Except as adjusted by the heuristics described in section 3.3.2, the interpretation of each character is the letter associated with the model that best matches the drawn marks. Sometimes, the user draws marks that are interpreted as a character other than the one the user intended, for one of two reasons. The

user could have drawn a badly formed character, one that more closely resembles a model for a character other than the one intended. Or, the models for the two characters that were confused could need adjustment. In the second case, either the models for the character the user intended to draw need to look more like the marks the user drew, or some model of the character the system thought the user intended looks too much like another model and needs to be changed or eliminated. Since our system can have multiple models for each character, it is possible for one or more of those models to be incorrect or extraneous.

Sometimes two characters are actually drawn very similarly by a user. For example, a lowercase 'l' might look very much like an uppercase 'I'. In these cases, the system must rely on reordering heuristics and easy correction methods to obtain good behavior. If the user draws two characters in the same way, there is nothing the system can do (looking only at a single character) to tell the difference between them.

Tuning the models can be done most easily with user intervention. It is the user who must make the determination between a well-formed and a poorly-formed character. Since the system is attempting to recognize the user's handwriting, it has used all its information about what those characters look like to make its guesses about which character the user drew; it has no other information about what a character "should" look like (although we discuss ways the system could get around this difficulty and tune itself in section 3.6.2 below).

We decided that any character that the system interprets correctly is probably represented by a good set of models; that is, the models for the character do not need adjusting. If the user corrects a character, then it is possible that the marks the user drew were actually a well-formed character and that the models need tuning. Since the user must make this determination, the system must supply two things: the marks the user originally drew and a way for the user to inform the system whether or not the models should be tuned.

To show the original marks the user drew, we simply pop up a small window containing those marks above a character when it is corrected. This is shown in figure 3-4. In addition we create a small button below the character that the user can push if the displayed marks look like a well-formed character and the user wishes to tune the models. This is less intrusive than many other methods because if the character was poorly formed or if there is no desire to tune the models, the user can simply continue entering text and the popped up window will disappear.

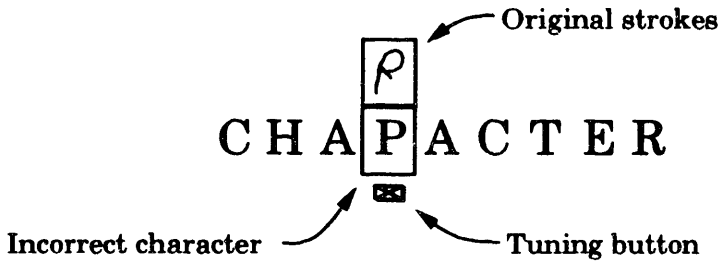

Figure 3-4: Showing original strokes in a character being corrected.

Experimentation provided several twists on this practice. It seems to be generally the case that most people's handwriting contains at least one pair of letters that are formed similarly. Much of the time, ambiguities between these letters can be removed using the heuristics described in section 3.3.2. In cases where the heuristics fail, the tap-for-second-choice gesture generally provides the correct interpretation without disturbing the user's flow of writing. In order to maintain this smoothness, we decided that the original strokes and model tuning button should only pop up for characters that had to be redrawn. These were characters for which there was a serious problem, since neither the first nor second choices were correct. Thus, in normal writing, there aren't original strokes and tuning buttons popping up constantly, but only in cases where gross misrecognition has occurred.

Another slight variation is based on the observation that people often want

to finish writing a word or phrase before tuning the model letters, even when they are sure they wanted to do this tuning. For this reason, the tuning buttons remain beneath redrawn letters even when a user is entering new text. The tuning buttons serve the double function of both allowing the user to tune the models and marking which characters were redrawn so the user can go back and tune them at any convenient point.

3.4.1 The tuning process

When a user presses a tuning button, the system pops up a special tuning window. A sample tuning window is shown in figure 3-5.

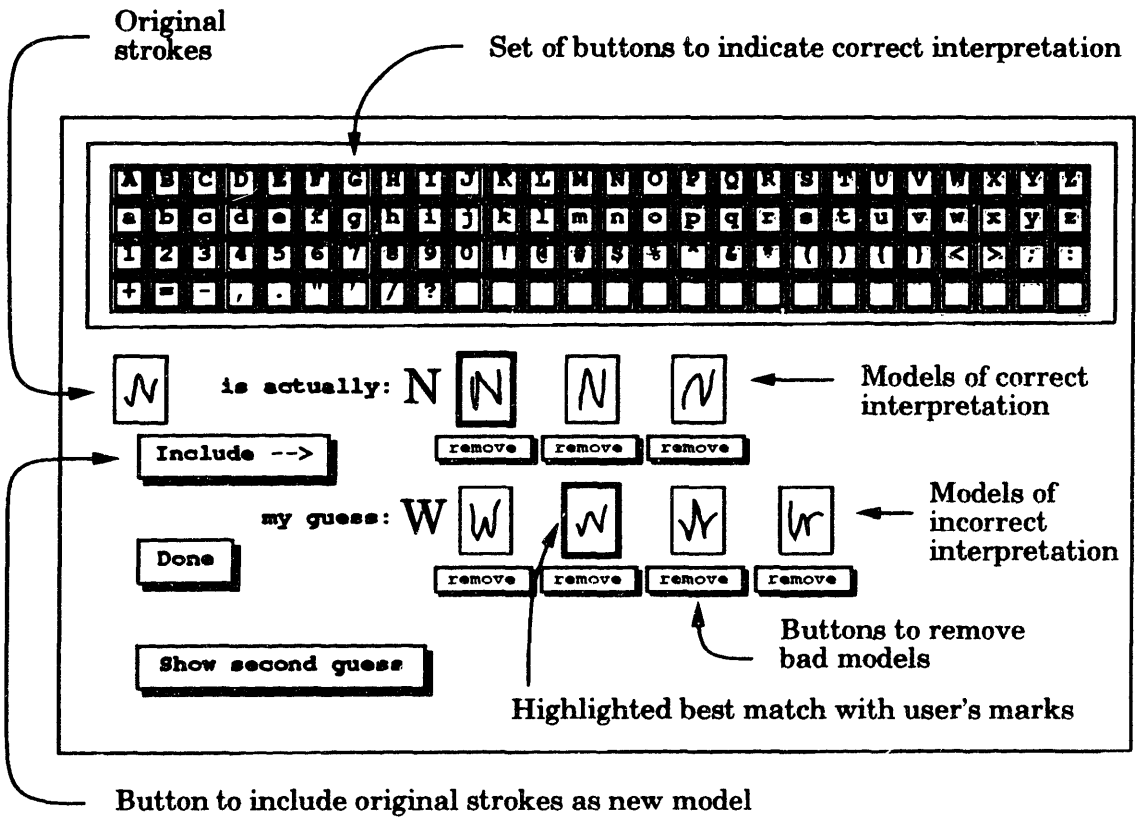

Figure 3-5: A sample tuning window.

We tried including various things in this window, and eventually decided on the following:

- The original strokes the user drew. The tuning window may have covered the input area, and the user may wish to compare the drawn character to the existing models.

- A set of buttons containing all possible input characters. If the system is unable to recognize a particular character, then the user must have some way of indicating which character was intended. When the user presses one of these buttons, the set of models for the character in that button is displayed and it is assumed that the character in the button is the character the user intended to draw.

- The set of models for the letter the system thinks the user's marks represents. If the user successfully corrected the character before pressing the tuning button, then this is the character the user intended. If the user has pressed one of the buttons described in the previous paragraph, then the system uses that character. The model that most closely matches the drawn strokes is highlighted.

- The set of models for the character that was the system's original interpretation of the user's strokes. The model for this character that most closely matches the drawn strokes is also highlighted. Along with each model, both of this and of the correct character, is a button that allows the user to remove the model, if it is not a good example of the character.

- Several other buttons. One button causes the drawn strokes to be included in the models for the "current" character. Either they are added as a new model or they are averaged into an existing model (if they resemble that model closely enough). Another button causes the models for the system's second

choice for the drawn strokes to be displayed, so the user can remove any of those that are incorrect. A third button dismisses the tuning window.

Although we were able to use this tuning method quite effectively, and quickly tuned our character samples to get very high accuracy, the tuning process is an additional burden on users. Some who were familiar with the character recognition process understood the concepts behind the models and were able to use the tuning window, but others found it confusing. Section 3.6.2, below, describes some ways that user-directed tuning could be performed automatically.

One thing that works well about this method is that tuning happens during text entry, so that it accurately reflects the characters users write, rather than the characters users wrote when initializing the recognizer. Another thing that works well about it is that the models' accuracy increases more quickly than if this were done automatically because the user decides which models need tuning. Also, since the user is only tuning the models for one or two characters at a time, the tuning is just a small inconvenience, but still has dramatic results.

3.5 Making the usable useful

All these pieces put together make for a reasonable system, but it is still certainly nothing one would want to use on a regular basis, even just for entering short strings. Some additional functionality makes the system much more useful. Users should not be limited in how much they can write and should be able to delete and insert text as they are entering it.

3.5.1 Writing more

One problem with a notebook-sized computer is that the screen is very small. Add to that the fact that in order to get reasonable recognition and to prevent hand cramps the user must write letters at least $\frac{3}{8}$ths of an inch high and it quickly

becomes apparent that there is not room to write much. In addition, it is desirable not to use the entire screen as an input area, as the user may need to view other material when entering text.

In a system with a keyboard, this is not a problem because the entered text can automatically be scrolled off the screen as a user types. This doesn't work with a stylus-based system, however, because the user directly positions the tip of the stylus in the place where the next character goes. The system can't scroll the entered text out from under the user's pen!

One effective solution is to add a small button marked "more" to the input area. When the input area is filled, the user simply presses the "more" button and the entered (large) text is pushed up out of the input area into a smaller font. The user can still read it, but the input area is now available for more text entry. This is analogous to pressing the "return" key on a typewriter. Figure 3-6 shows some text that has been pushed out of the input area and some additional text that has been entered after the more button was pushed.

The user must also be able to retrieve that text, in case it contains errors, and edit it (see below), so the smaller text takes the form of a button. When the button is pushed, the small text moves into the input area (where it can be manipulated) and the current contents of the input area are pushed to below, as in figure 3-7. This allows users to write full-sized letters, yet still enter a reasonable amount of text.

3.5.2 Deleting text

As people write, they often edit their text. Sometimes a user will accidentally write an extra character. Sometimes a user will misspell a word. Sometimes it may be that the text isn't pleasing and an entire word or phrase should be removed. When writing on paper with an ink pen, one simply crosses out the extraneous material. This is a natural gesture with a stylus too.

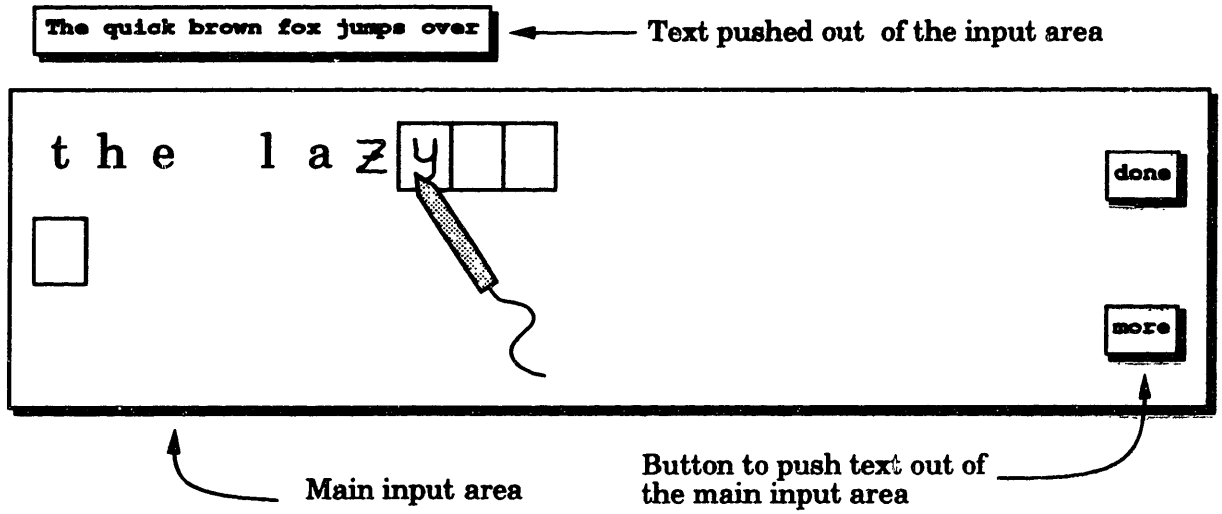

Figure 3-6: Some text pushed out of the input area with the "more" button.

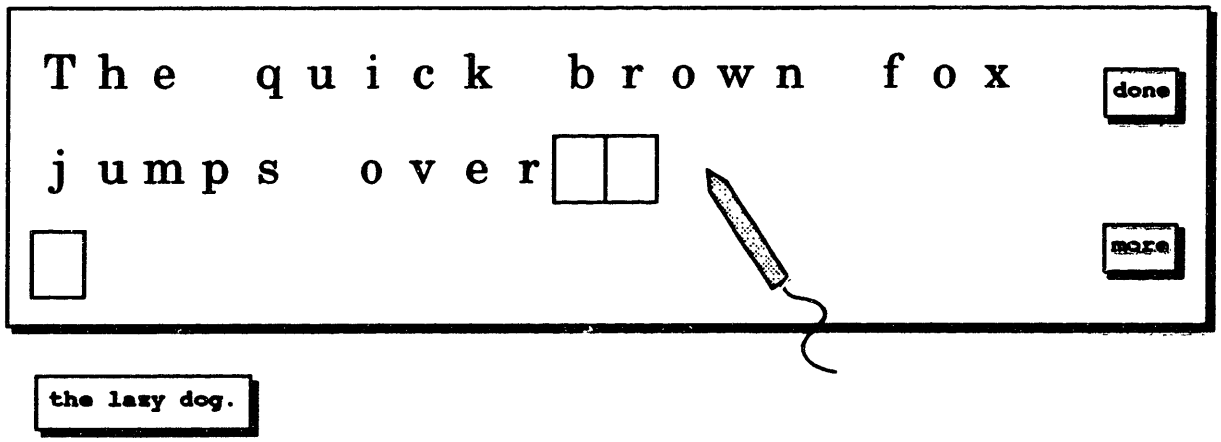

Figure 3-7: Text pushed below the input area.

Consequently, we implemented two further editing gestures (beyond the tap-for-second-choice gesture). One is the delete character gesture (a vertical stroke through a character), and the other is the delete word gesture (a horizontal stroke through several characters). These gestures are easily distinguishable from new text because they occur over characters that have been previously entered and interpreted. The delete character gesture removes a single character and closes up the space it occupied – it could be used to correct "coat" to "cot." To replace a mis-entered or misrecognized character, a user simply taps twice and redraws it – for example, to correct "cot" to "cat." The delete word gesture actually deletes any contiguous string of characters.

These gestures proved easy to use and easy to understand and test users got the hang of them quickly.

3.5.3 Inserting text

Often text needs to be inserted. In some cases the text to be inserted is a single character, such as to correct "cot" to "coat." Here the system should open up only a single character space for the new character. In other cases, however, the user may want to insert a whole word, or even as much as a sentence, and needs a way to open up an unlimited amount of space. Rather than having two gestures to remember, we formulated the following scheme. If a user makes a caret between two characters, a single character space opens up and the segmenting cursor moves to that space so the user can enter the new character, as shown in figure 3-8. If the user makes a double caret, however, (by making a second caret to the left of the space opened up by the first caret) we split the text in the input area in half, pushing half of it upwards, as if the user had pressed the "more" button, and half of it downwards, leaving the input area free for the insertion of an arbitrary amount of new text. This is shown in figure 3-9.

Because these gestures (the delete marks and the insertion caret) are similar

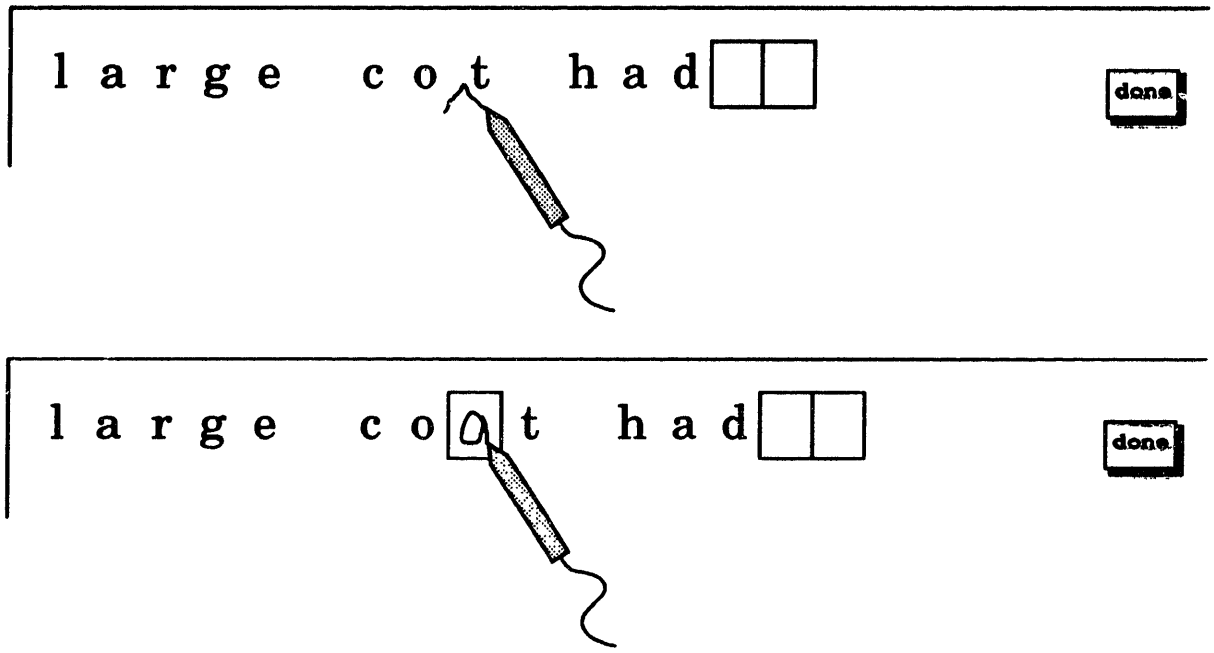

Figure 3-8: Inserting a single character

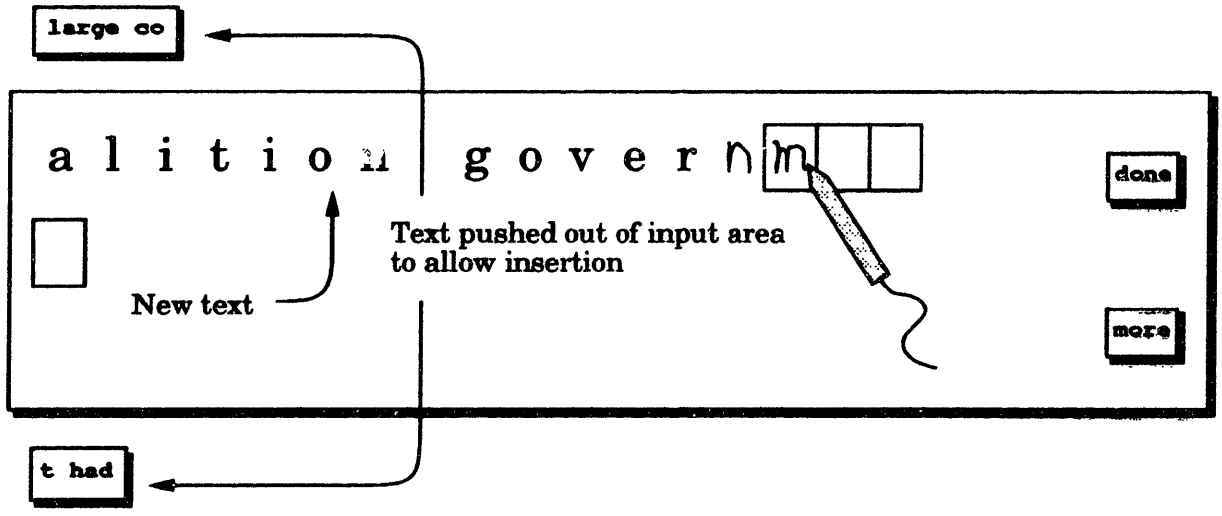

Figure 3-9: Inserting more text

to common proofreading marks, our test users found them easy to understand and make. These marks are natural for editing text as it is being entered. A more detailed study of text editing is made in chapter 4.

3.6 Future improvements

This interface for entering text is fairly effective. Several members of the research and secretarial staff tried it (as our test users) and found it reasonably simple to learn and use, even with various amounts of familiarity with the project.

Although the system worked reasonably well, there are a number of improvements that seem feasible and that we did not have time to implement fully. In particular, it would be nice if users were not required to write in boxes, and if the system were capable of tuning its model letters itself, without user intervention. In addition, the system could possibly supply ways to reduce the amount of text users must actually write.

3.6.1 Getting rid of the boxes

The segmenting cursor dramatically reduces the sense that one is entering text into boxes on a form. Nonetheless, it is desirable to eliminate the necessity for boxes altogether. Some experiments were performed along these lines with the recognition software and it seems feasible, although somewhat more computationally intensive, to recognize unsegmented characters. There are other difficulties, however.

It is very clear to almost all users, when they are writing in boxes, that each character must be written in a separate box. Almost no users wrote letters that were connected. Without boxes, however, users tend to write certain familiar groups of letters as a single character, such as 'th' or 'ing'. Even when instructed specifically not to connect adjacent characters, users were unable to prevent this.

The problem was even more dramatic when they were using a pen and paper. So as stylus and screen hardware improve, this problem will become more pronounced. Rather than attempting to prohibit this behavior, we propose that it be allowed (or perhaps encouraged) and that these handwritten 'ligatures' be included in the set of user characters. It is unclear how well this will work, as users may begin to string all their characters together, but it is an approach that bears investigation.

3.6.2 Tuning without user intervention

Another inconvenience in the system as it currently stands is the process by which the recognition system is tuned. A user must understand some key concepts about the recognition system in order to tune it, such as the fact that it recognizes characters by matching them against models and that the system maintains multiple models of each character. It might be possible, however, to tune the recognition models automatically without user intervention, or to tune them with a simpler form of user intervention.

The recognition models need to be tuned while the user is writing normally, because users do not seem to have a good sense of how they actually form characters. When users were asked to write a set of sample characters, forming them normally, they made characters very different from those they made in regular writing.

One way to tune the model characters without user intervention is to use heuristics to identify good and bad models. One such heuristic might be as follows: If a model is used to identify a character that is subsequently corrected to a different character, then that model is marked as potentially incorrect. If a model is used to identify a character that is not corrected, then that model is marked as likely being correct. If a model receives a large number of "incorrect" marks, but few or no "correct" marks, then it is likely a poor model and should be removed.

Another heuristic, which is more dangerous, might be used to improve the

models. A character that is written, identified, and not corrected is possibly a good model and could be included (or averaged) into the set of existing models. This is dangerous because the character could have been misrecognized, but the user could have missed the error. In this case, the set of models could become worse by the inclusion of an erroneous character.

Alternately, some input from the user could differentiate between well-formed and poorly formed characters. The system could periodically prompt the user to decide which of two characters was better formed; preferably some time when the user was not in the middle of something else. To make this less intrusive, the prompting could be done in a small, nearby window that the user could ignore. It is not clear how this could be done most effectively, but it is a good area for further study, because we found incremental tuning of the recognition system to cause dramatic improvements in recognition rates.

3.6.3 Writing less

Another problem with our current system is that, at its core, it is a tool for text entry by the laborious process of hand printing each character. Allowing a user to write in some form of script might make the system more natural to use, but would probably not be a significant improvement; users would still be required to write out all the text they wished to enter. Modifications to the system to reduce the amount of writing necessary would make text entry significantly faster. The system could complete words partially written by a user, predict the text the user was about to write, or provide some method of quickly copying text to the insertion point from some other place in the document.

Completion

Many screen editors have a word completion feature. The user types part of a word, presses the word completion key, and the editor searches for a likely

completion for the word, either from the text already entered or from a dictionary. This technique could also be applied to stylus input. We did some very simple simulations, however, and found that the savings (in number of characters still necessary to draw) was almost always insignificant, especially considering the extra work necessary to call up the completion.[9]

Prediction

Another method for reducing character entry is that described in a paper by John Darragh, Ian Witten, and Mark James ([12]) about a system called *The Reactive Keyboard.* In this scheme, the system predicts what characters the user will type after each entered character and the user has the option of accepting any number of those characters or entering another character. An example of this scheme adapted to a stylus-based system: At some point the user might have written the string "A stylus-ba" in the process of entering the sentence "A stylus-based user interface is very useful." The system at this point might be predicting that the following characters were "sed user interface could be used," based on what the user had entered earlier in the document. The user could accept the predicted characters as far as they were correct, and then hand print the rest of the sentence, for a significant savings in entered characters.

We did two tests along these lines. One was a statistical simulation of this sort of prediction over a number of textual documents. In general, if we assumed that the user would only accept the next few characters if the first five or more of them were correct, then approximately 30% fewer characters needed to be entered, depending on the sort of document. In documents with a significant amount of repetition (for example, legal documents), this percentage could be higher.

[9]The simulation consisted of scanning a text document and attempting to complete words of more than five letters, using only the first three and searching backwards in the document for the first match. Obviously there are other, more sophisticated completion methods that might be more effective.

The other test was to bring this type of prediction into the existing text entry system. We tried this by floating a small 'button' containing the predicted text next to the segmenting cursor. As the user writes, the text in the button changes to correspond to the next prediction. If the user wishes to accept part of the prediction, it is necessary only to press the button at the point in the text where the prediction is correct. In the above example, therefore, the user would have entered "A stylus-ba" and the button floating next to the final 'a' would contain the text "sed user interface could be used." If the user pressed on the button over the second 'e' of "interface," then the text "sed user interface" would be included into the entered text, as if the user had written it by hand.

This is not particularly effective in our system because the prediction is based on the previously entered text, and in order to get reasonable prediction, a fair amount of text must be entered. Since we were using the system to enter small amounts of text, the prediction was not that good. This does not mean that this method could not be used effectively; the prediction might be good for entering text in the middle of large documents, since the prediction could be based on the rest of the document. In fact, we did a statistical simulation of this, in which we removed a single sentence from a document and then attempted to predict that sentence. In general, the results were almost the same as predicting regular text being entered: a reduction by about 30% of the entered characters. Although this is not phenomenal, it is certainly worth considering.

Our "floating button" method is probably not the most effective way to present the predicted text. One problem with it is that the button must be to the right of the entered text, since that's how the text would be entered, but in that position the user's hand sometimes covers the button. Another problem is that it is too much trouble to decide whether the predicted text is sufficiently correct to switch from entering characters to accepting part of the prediction. This might be a problem in any prediction system, or it might be eliminated if the predicted text

were presented differently. In any case, text prediction could be used to reduce the amount of text that actually needs to be entered by hand and should be given further study.

Quick copying

A third method for reducing the number of characters entered is to give users some quick way to copy existing parts of the document. For example, if a simple gesture indicated that a word should be copied from its position in the document to the insertion point, then a user could make a gesture to copy a word instead of hand entering it if the word were easily accessible in the document. We did not have the opportunity to experiment with techniques of this sort, but they could be useful.

3.7 Summary

There is much more to text entry with a stylus than recognition of hand printed characters. Users must be able to edit the text they are entering, and this is facilitated by nearly immediate feedback of the system's interpretation of their writing. Users should also not be restricted to entering only the amount of text that will fit into a window. Since recognition systems are imperfect, users must be able to correct mistakes. Tapping on errors either to correct them automatically or to re-enter a character is an easy way for users to handle misrecognition.

In addition, text entry systems are complicated by technical issues. A user's marks must be segmented into characters; the segmenting cursor provides a way to accomplish this without presenting the user with rows of boxes. And the ability of the system to recognize hand entered text can be dramatically improved if its models of each character are tuned during the text entry process.

Our implementation of these ideas provides a good system for text entry, but there is always room for further study.

Chapter 4

Text Editing

Although the previous chapter describes ways in which a stylus-based system can be used for text entry, a keyboard is usually better suited for this task. For the task of text editing, however, a stylus-based system is better than most. A stylus is well suited to the type of interactions necessary for text editing, and many of the situations in which a stylus-based system would be useful involve text editing and manipulation.

But how should a stylus be used for text editing? Text editing with a mouse consists primarily of selecting regions of text and operating on them by pressing on-screen buttons or selecting options from menus. With a stylus, however, a user can draw marks on the text to edit it, much like editing on paper; the system can utilize a "gestural" interface.

This brings up the issue of what gestures the system should understand. Users must not be required to remember too many gestures and the gestures must be sufficiently different from each other that users can tell them apart. Plus the system must be able to recognize the gestures. This is a different problem than the task of recognizing characters, because gestures can be made in many sizes and orientations.

Then there is the question of how the input of new text should interact with

the editing process. Should text entry be integrated with editing? Or should it be a separate mode? Our system uses modal text entry, but we discuss both options.

Despite the use of gestures for most editing tasks, it is still necessary to select regions of text. There are many different ways to select a region with a stylus, such as circling it or marking its ends, and our system allows several of them.

In the process of building our system, we asked several test users to perform certain editing tasks to help us ascertain the usefulness of various features and generate ideas for improvements. With their feedback, we made some additions to the system to make the editing process more natural.

4.1 Making a case for a "gestural" interface

There are a variety of ways to use a stylus for text editing. It could be used like a mouse to mark regions and occasionally, with menu selections, to perform common editing operations such as copying and moving. What seems to make more sense, however, is an interface based on drawing marks. A stylus is similar to a pen; when a user puts the tip of the stylus on the screen surface, it leaves a mark. Because this is an operation with which many people are familiar, it makes sense to use this sort of interface to communicate with a computer system.

An interface in which commands are communicated to a computer by drawing lines with a stylus is known in most of the literature as a "gestural" interface. Of course, it does not involve most gestures, but since it is a common term, we will use it here.

A gestural interface makes sense because it takes advantage of the characteristics of a stylus. Like a mouse, a stylus is good for indicating a position on a computer screen. Rather than moving a cursor to an area to be edited, a user can make the appropriate editing mark directly on the text in that area.

Unlike a mouse, it is easy to draw with a stylus. Thus editing changes can be

communicated to the system by drawing some representative gesture rather than keying in a command or dragging through a menu tree to an appropriate selection.

4.1.1 "Chunking"

In the design of a system with gestural commands, it is desirable that a command correspond to an operation in the mind of the user issuing it. For example, if a user wants to delete a region of text, the necessary action should encompass that entire command: the location and extent of the text, and the fact that the text is to be deleted. The user should not need to mark the beginning of the text using one command, the end of the text using another command, and then indicate that the text is to be deleted with a third command. There should not be several separate commands to perform a single action.

It is necessary to make tradeoffs, however. If each action and all the associated parameters were indicated with a single command, then there would be too many commands to remember. But common commands can be designed so that the desired result is expressed in one action by the user. That is, the "chunk" of action that occurs when a user makes a command should correspond to the "chunk" of action that the user is trying to accomplish, at least for common commands.

Gestural editing is well suited to this requirement. Because editing marks can be drawn directly on the text to be effected, they can include the location and extent of the text as well as the desired change.

4.2 Choosing gestures

Once we have determined to use a gestural interface for text editing, we must choose a set of gestures. They must be different from each other and familiar enough for users to remember.

The gestures must be different from each other for the benefit of both the

users and the system. If two gestures are similar, a user may have difficulty remembering which one has which function. There might also be a problem with people drawing a gesture that looks too much like another gesture. Gestures that are very different from each other are also easier to distinguish during the recognition process (described below in section 4.3).

The gestures should be familiar to users. The gesture for a particular function should be what a user expects to draw to indicate that function and the function of a gesture should be the one a user would expect. For these reasons we chose gestures that are simplified versions of common proofreaders' marks. We identified a set of simple editing commands that seemed useful in an experimental system and chose gestures for those commands. The functions we chose include: delete character, delete word(s), insert, and transpose, and their gestures are pictured in Figure 4-1. The system actually understands several other gestures, but they are described later in sections 4.5 (on selection) and 5.2.2 (on undoing).[1]

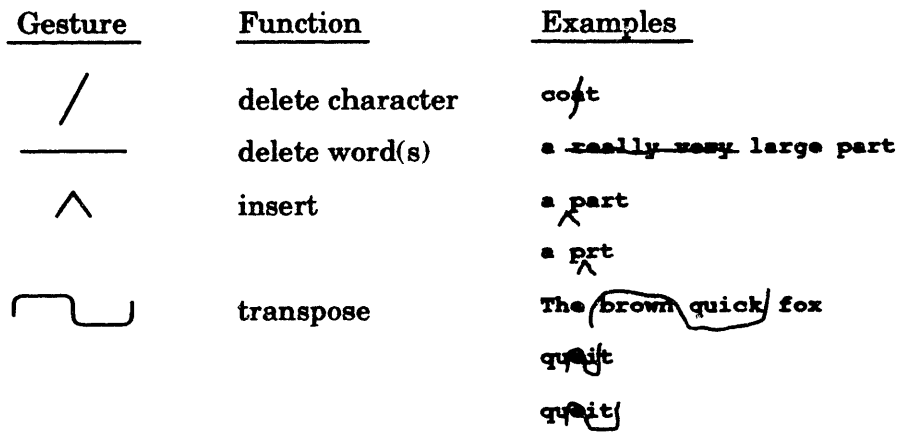

Figure 4-1: Some of our gestures for text editing.

We chose these marks because they are similar to common proofreaders' marks,

[1]These gestures are similar to those described in *FIDS – A Flat Panel Interactive Display System* ([20]) and *A Gesture Based Text Editor* ([36]). Both of these works also use simplified proofreaders' marks.

but much simpler, and because they are very different from each other and, consequently, easy for users to remember, easy for users to make, and easy for the system to interpret.

Actual proofreaders' marks are not well suited to gestural editing. Because they are designed to communicate clear corrections on pages set with lead type, proofreaders' marks include many extraneous strokes. For example, the mark for deleting a character is a strike through the character (indicating that it should be removed), and two other marks indicating that the space the character occupied should be closed up. In a computer system, however, the space occupied by a deleted character should almost always be closed up, so this should not require a three stroke gesture. With our gestures the common case can be communicated in one stroke, and the uncommon case, in which the character should be removed and a space left in its place, can be communicated in only two (a delete gesture followed by an insert or a double tap).

Most proofreaders' marks suffer from similar unnecessary complexity. And many of them consist not only of marks on the text, but several character descriptions of the required changes written in the margin. To delete a phrase, for example, a proofreader marks the phrase as it occurs in the text, and then writes "del." or some similar abbreviation in the margin. Stylus-based editing has the capacity to be much simpler than this, so gestures can be based on proofreaders' marks, because many people are familiar with them, but should not be those marks exactly.

4.3 Recognizing gestures

When a user makes a mark on the screen with the stylus, some method must be used to interpret that mark. This is not as constrained a problem as the interpretation of characters, because some gestures can be formed in a variety of

styles, sizes, and orientations, all by the same user. In addition, even if a gesture has been identified as belonging to a certain class (e.g. a delete gesture), the system must still identify the parameters to that class (what to delete).

4.3.1 Nonsensical gestures

One issue that came up early in this exploration was the question of how the system should react to a gesture it cannot identify. We came up with two reasonable responses: the system can ignore the gesture or the system can do whatever it thinks makes the most sense. There are justifications for both of these.

If the user makes a mark that the system does not understand, then the system could ignore it. This prevents the system from interpreting the user's gestures in some way other than that which the user intended. If the user thinks that the system understands a class of gestures that it doesn't, the system can signify that simply by not responding to the gesture.[2] If the user is doodling, then the system is likely not to do anything destructive, since it will ignore most marks.

On the other hand, it is unlikely that a user, especially one with some experience on the system, will make a mark that is not intended to be a gesture that the system understands. That is, users trying to get work done will almost always make gestures intended to have some function. The system, therefore, should not ignore any gestures, but should do its best to interpret them.

In our original implementation, we used the latter approach. Any mark was interpreted as a valid gesture belonging to one of the recognized classes and the system went on to attempt to determine the parameters to the command that class of gestures indicated. What we discovered, both in using the system ourselves, and in observing others attempting to use the system, was that the system sometimes misinterpreted gestures. Because our system included the ability to undo edits (see

[2]The Penpoint system ignores nonsensical gestures, but flashes a question mark to indicate that the gesture was not understood.

below, in section 5.2.2), this was not a major problem, but the system attempted to interpret even completely nonsensical gestures and this seemed strange.

Our first conclusion, then, is that the system should respond in a predictable way. The interpretation of a gesture should make sense. If a human observer would not put a gesture into a particular class, then the system should not either. The current system, therefore, is fairly lax about the gestures it accepts, but we ensure that the most obviously meaningless gestures are rejected.

4.3.2 Our recognition method

The method we use to classify gestures is very similar to that described in Dean Rubine's PhD thesis ([27]).[3] First off we restrict ourselves to gestures consisting of a single mark. No gesture requires that the user make a mark on the screen, lift the pen, and then make another mark. In addition to making the system easy to use, this simplifies the recognition task.

While the user is making a mark, the system does nothing but show the mark on the screen while recording in memory the path traced out by the stylus tip. When the user lifts the stylus off the screen, the system erases the mark and then analyzes the stylus' path. After the path is smoothed, the values of several statistical functions are computed. These values are then fed into several tests, each of which returns a Boolean result. A gesture class is described as a mask (signifying which tests are important to identify the class) and a set of values (what the results of those tests should be).

We determined by hand which functions to compute, which tests to use, and what the values of those tests should be for each gesture class we wanted to describe. Dean Rubine's thesis describes a more systematic way to specify gesture classes (although his system requires training by users) and incorporates more functions. Appendix A includes tables of the functions we compute and the tests

[3] Our simpler version is inspired by a draft of his thesis and was implemented separately.

we perform, along with the gestures we interpret and the masks and values for those gestures.

4.3.3 Interpreting gestures

Once a user's mark has been classified as belonging to a particular gesture class, the system still must determine the parameters to the command that the gesture indicates. For this we built algorithms for each gesture based on the assumption that the user is editing English text. As an example, consider a delete word gesture. Since that gesture is intended for deleting one or more words, it makes sense to extend the region deleted to include entire words. Since a user is likely to make a mark at least most of the way through a word, however, the system should also restrict the region deleted so as not to include those words into which the user's mark does not sufficiently penetrate. This is shown in Figure 4-2.

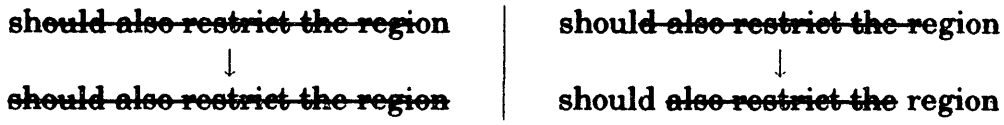

Figure 4-2: Using heuristics to interpret delete gestures. If a user makes a mark such as those in the top row, the system interprets them as shown in the bottom row.

Although the addition of these heuristics improves the system's behavior significantly, it is clear that more powerful heuristics are desirable. The system should have as much knowledge of the language being edited as possible. This harkens back to the discussion of "chunking" in section 4.1 above. Wherever possible, the system should use its knowledge of the task to interpret a user's mark in what is most likely the way the user intends it.

In our observations of people using the system to make some changes to a document (described in section 4.6), we noted that no users mentioned the presence

of these heuristics. Indeed, they only noticed that something unexpected happened when our heuristics were not sufficiently sophisticated to divine their intent. Most of the time, the user's marks were not precise enough to be interpreted correctly without the heuristics, but with the heuristics, the system behaved as desired.

4.4 Input while editing

During the editing of a document, there are various tasks that necessitate entering text. The most obvious of these is adding new text to the document, but there are others, such as specifying a string for which to search a large document, or a file name in which to save the document.

4.4.1 Modal input

The simplest way to implement text input while editing is by having a special "mode" for text entry. Text entry is usually not allowed; a separate mode presents an opportunity to enter text, possibly restricting what else the user can do. For example, if the user signifies that some text is to be inserted in the document, the system could pop up a special window in which the user can enter the new text. This is simple because there is no interaction between the marks a user makes during text entry and the marks a user makes during text editing.

This also has the advantage of allowing the text entry process to take up a significant amount of screen real estate, since the user won't be using the screen for anything else. If text entry were to be integrated with the rest of the editing process, then it would not be able to use as much of the screen, since a user would need to be able to perform whatever other tasks were in progress. Since text entry requires that the user write fairly large letters, it makes more sense for text entry to have access to a large portion of the screen, especially on a notebook-sized computer.

Another benefit of modal input is that the clear distinction between the process of editing text and the process of entering it allows the system to use multiple text entry techniques. Sometimes a user might enter text by hand printing it; other times the user might enter text by tapping letters in the image of a keyboard on the screen. Other parts of the system would not need to be concerned about what method was being used.

The system we constructed employed modal text entry. There was a clear distinction between the process of entering text and the process of editing it. Users who had had experience with the text entry described in chapter 3 were able to use the skills they had developed with the editing system. They did not have any trouble distinguishing between those actions that were allowed during text entry and those that were allowed during text editing, because they were presented with a distinct display during each of these modes.

4.4.2 Non-modal input

On the other hand, it might be desirable to integrate text entry with text editing, especially for purposes of adding new text to a document. This adds significant complexity. Such a system must not have significant overlap between the gestures used in the text entry process and those used in the text editing process, because a gesture should have only one meaning. If text entry and text editing are integrated, there should not be any gesture that means one thing when entering text and something different when editing it.

For example, it might be desirable to use a single tap gesture in the text editing process. This would be dangerous because the single tap gesture is used to correct misrecognitions during text entry. If this gesture had some similar effect while editing, though, it might be possible to use it for something slightly different, such as checking and correcting the spelling of a word, which has a similar effect to correcting a misrecognition: changing the text from incorrect to correct.

Non-modal text entry might make more extensive use of the segmenting cursor. In standard, keyboard-based text editors, a cursor signifies, among other things, the point of insertion. Instead of having a cursor the size of the text being edited, the line of text containing the insertion point could have a large, segmenting cursor instead, and a user could write directly into this cursor. This idea is shown in figure 4-3.

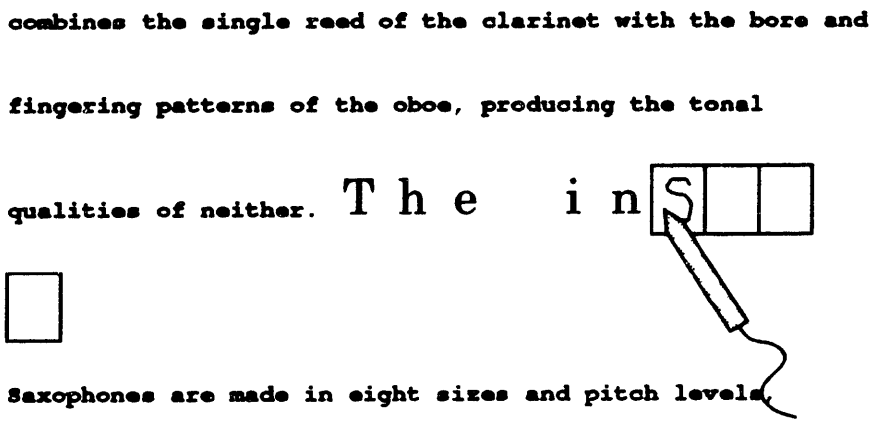

Figure 4-3: Using the segmenting cursor instead of a regular cursor in a text editor. This would allow text to be entered directly into a document being edited without necessitating a text entry mode.

We were not able to experiment with this idea, but it shows some promise. Of course, for some things, it might still be desirable to pop up text entry windows (specifying file names, for example), but the use of the segmenting cursor could blur the distinction between text entry and text editing.

4.5 Selection

Although very common commands, such as deleting and inserting text, can include the scope of the command in the gesture that specifies the command, this is neither possible nor desirable for all commands. For example, the command to move a

paragraph from one section of the document to another will be used rarely enough that having a special gesture for it would not be practical. Users would not be able to remember the gesture, and, because there would be so many different gestures, it would probably be complex and difficult to draw.

Instead, as in many editing systems, a region of the document can be selected using one command and then various other commands can be used to operate on the selected region. An issue in a gesture-based editing system, therefore, is how to use a stylus to select a region of text.

One method is circling. This especially makes sense when the region of text to be selected is small, or roughly circular (like a paragraph or element of a table). Users commonly use a pen to circle things on paper and it is natural to do the same on a pen-based computer. We therefore include the circling gesture in our system in addition to those described in section 4.2. This gesture is easy to identify using the recognition method described above, and it is also easy to interpret.

Circling is an excellent method for selecting several words all on the same line. It is awkward, however, for selecting oddly shaped regions of text, such as a phrase that is broken across two lines, or a sentence out of the middle of a paragraph, Figure 4-4 shows two different regions of text being circled.

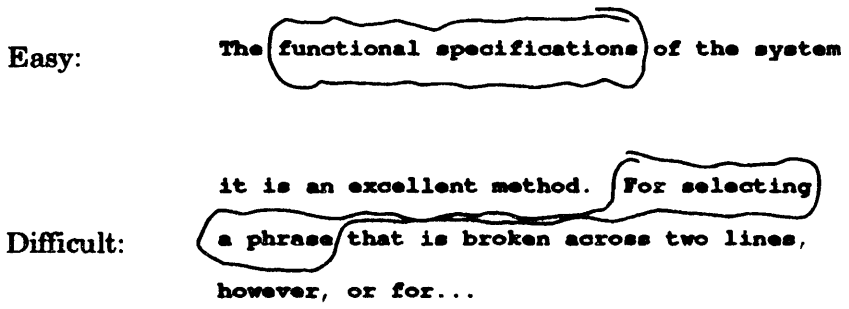

Figure 4-4: Circling text. This is easy for small regions of text, but many regions of text are awkwardly shaped.

An alternate method of selecting text is to mark the beginning and end of the

region. In a keyboard based system, this is often done by moving the cursor to one end of the region of text, marking it with a special keystroke, then moving the cursor to the other end of the region. This can be slow and time consuming. With a stylus, however, this operation is simpler, since indicating a point in the document requires that a user merely point the stylus there -- much faster than moving a cursor.

We implemented this selection method by including two new gestures for selecting a region. One gesture indicates the beginning of a region and looks like a left bracket. The other looks like a right bracket and indicates the end of a region. They were simple to recognize using our established methods.

The interpretation of these gestures is interesting. When a user draws a circling gesture to indicate a region, it is clear that this region is distinct from any other region that has previously been selected. Since the bracket gestures are used to mark the beginning and end of a region, they could conceivably also be used to extend or restrict a region. Thus, if a user makes a left bracket before the beginning of an existing region, then it is possible to interpret this gesture as extending the existing region to include more text at the beginning. If the left bracket is within the existing selection, then it could mean that the user intends to shrink the current selection so that it begins at the point where the left bracket was drawn. If the user makes a left bracket past the end of an existing selection, however, it is clear that the gesture indicates a new region. A similar interpretation is used for the right bracket, or end of region, gesture. An example of this type of interaction is shown in figure 4-5.

These gestures were easy to use, both to select an awkwardly shaped region of text and to extend or restrict an existing selection.

Another method that can be used to make selections is "wipe-through." In a single gesture, the user draws a mark beginning at the beginning of the region and extending to the end of the region. This gesture is possibly faster and easier

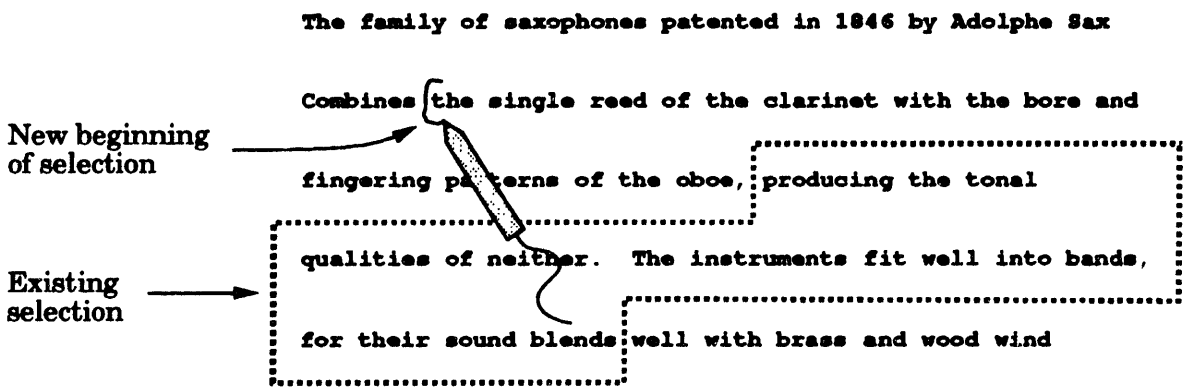

Figure 4-5: In this example, the selected region will be extended so that it begins where the left bracket mark is made.

to make than either the circling or bracket gestures.

The problem with wipe-through is that the gesture does not have any distinguishing shape. Not only does this make it difficult to identify, but allowing a wipe-through gesture for selection conflicts with having any other gestures. Because of this difficulty, we decided not to implement wipe-through selection in our system. The Penpoint system implements wipe-through selection by having a user hold the stylus in one position for a short delay, and then begin moving it. Our simple recognition method was not able to recognize this gesture, but the method described in Dean Rubine's thesis is.

Another way in which wipe-through selection could be made feasible, however, is if the stylus had a button on it (in addition to its tip). Such a button could be used to distinguish between regular editing gestures and wipe-through selection. For example, when a user held the stylus normally, it could be used for text entry and normal gestural editing. When the user held the stylus with the button pressed, it could be used for selecting text.

It is not clear, though, whether this is an effective way to use such a button. We had the opportunity to use several styluses with buttons on them and found it difficult to write and draw while holding a button down. And if the button is used

to toggle between interaction modes, it might confuse users.

4.6 Observation of various users

To get a wider variety of opinions, we asked several people with various backgrounds to participate in informal user studies. These people all had some experience using computers, but varied widely in their understanding of the issues involved in a system such as this one because of their different backgrounds. Some were researchers in related areas, some were researchers in unrelated areas, and some were support staff.

For purposes of evaluating the systems we built, we sometimes invited these people to use the system without instructions about any specific task. We let them experiment with the system, while giving some explanation of the system's capabilities. At other times, we asked them to perform a particular task.

In the case of the editing system, the task we asked them to perform involved making several changes to a small document. The system presented them with a paragraph containing some errors and a list of instructions about the changes to make, all on the computer's screen. The paragraph contained five spelling or typographical errors to be corrected in various manners. In addition they were asked to transpose two words, delete an extraneous phrase, insert a missing word, and remove a sentence. The document used for this test is included as appendix B.

When they were presented with this task, most of the users had previously used the system at least once to get the feel of the editing gestures. For the most part they did not have much trouble completing the task and used the editing gestures as we had anticipated. There were some problems with the parallax between the tip of the pen and the computer's display screen (because of the intervening surface used to track the pen), but once users overcame these difficulties, they were able to perform our editing gestures easily.

They reported that the gestures are easy to remember (there are only a few) and that they were sensible. One user even asked to be allowed to attempt the system without any description of the editing gestures and was successfully able to guess most of them (although the system was not intended to be used without at least minimal training). Users found that, for the most part, the system interpreted their marks as they had intended, and we noted that this was primarily because of our interpretation heuristics as their marks did not precisely correspond to their intents.

4.7 Making the usable useful

As with our text entry system, this editing process seemed to work well, but was missing some desirable features. We did not have the opportunity to extend it into as complete an editor as we would have liked, but we did experiment with two relevant user interaction techniques: scrolling and on-screen buttons.

4.7.1 Scrolling

We chose to experiment with scrolling because it is a general technique for moving around in a document, and some such technique was necessary to explore the editing of documents larger than one page. The question then arises whether scroll bars should behave at all differently in a stylus-based system than they do in a mouse-based system. We believe that they should.

There are many different features found in scroll bars in existing systems. Essentially, these features allow, in some manner, the following actions:

- Skipping to the next or previous page,
- Moving to an arbitrary position in the document,
- Scanning rapidly through the document, and

- Repositioning the current page with respect to the screen.

To perform the last three of these actions, a user can move a "thumb" in the middle of the scroll bar.[4] In most systems, moving the thumb downward causes the displayed area to move downward (farther towards the end) of the document. This has the visual effect of moving the text on the screen upward.

On a system with a keyboard and mouse, this is perfectly reasonable behavior. We quickly discovered with a stylus, however, that it seems strange to put the stylus onto the screen, move it downward, and have the text move upward. Because of the close link between the stylus and the screen, a user expects correspondence between the movements of the stylus tip and the information on the screen – such a correspondence is the basis of this sort of system.

Consequently, we experimented with a scrolling system in which the text on the screen moves the same direction as the stylus. We call it "direct scrolling." It is extremely well suited to repositioning the area of the document being displayed, although in this system it is the document that is being repositioned, not the screen. This is a principal difference. Instead of focusing on the document as a fixed object and moving a virtual window on it, this type of scrolling views the user as fixed, and allows the document to be moved through the field of view. Thus the user can directly manipulate the document, as is desirable in a direct manipulation system.

We did not have the opportunity to implement a complete direct scrolling system, but only a part that allows local repositioning of the document with respect to the screen. In the part of the system we implemented, the displayed text moves directly with the stylus; if the stylus begins at the middle of the screen and moves to the top (in the direct scroll bar), then the text at the middle of the screen slides to the top along with it. Figure 4-6 shows two examples of how we envision a complete system. Direct scroll bars are much like conventional scroll bars, but

[4] A "thumb" in a scroll bar indicates the user's current position in the document.

with the focus on moving the document rather than the screen.

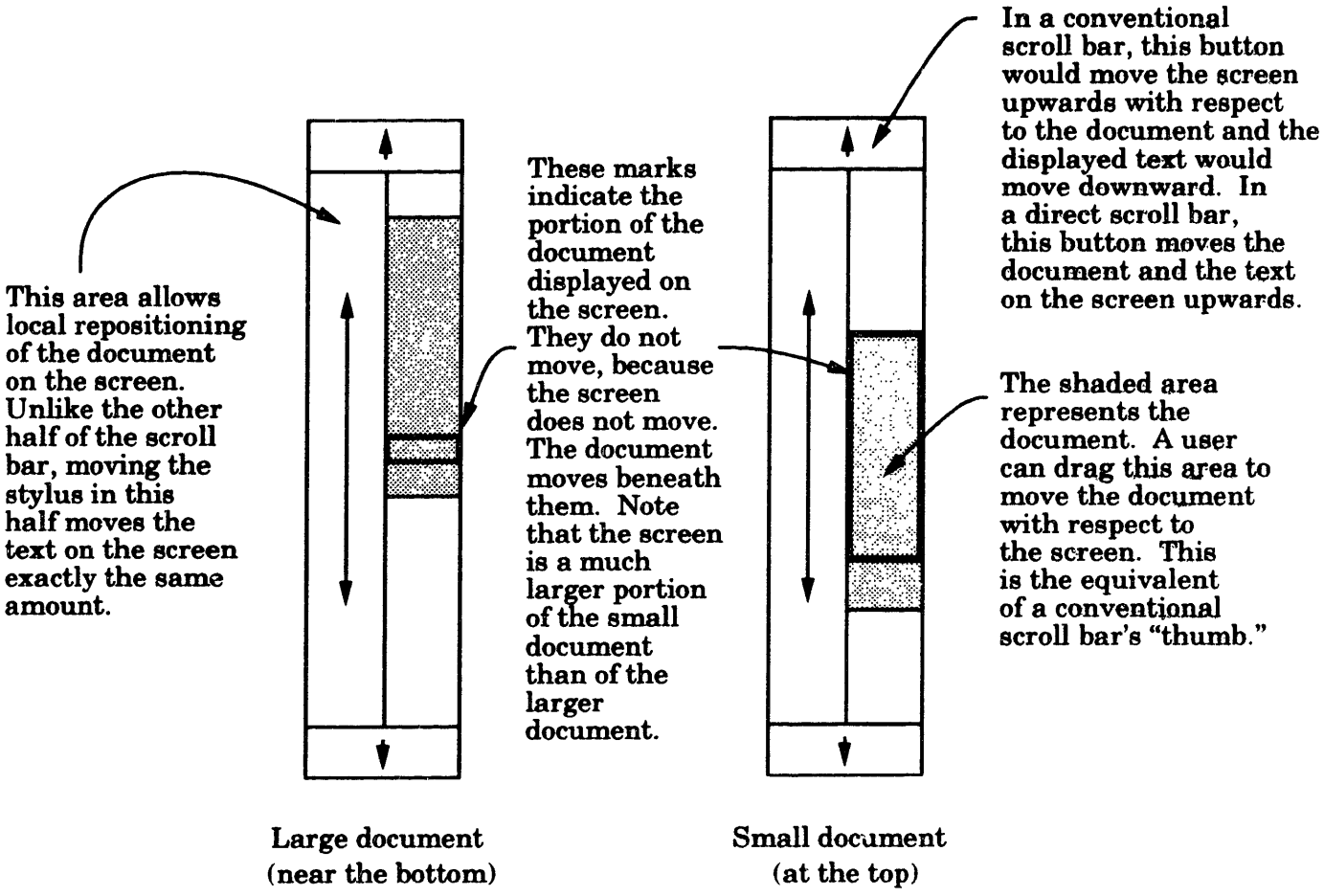

Figure 4-6: Direct scroll bars focus on moving the document rather than the screen. This figure shows examples of direct scroll bars for two different sized documents. Direct scrolling is better suited to a stylus-based system than conventional scrolling because the text on the screen moves the same direction as the tip of the stylus.

4.7.2 Buttons

Another common virtual device is the on-screen button. In principle, these are very simple: a button has an action associated with it, and a user positions the cursor

over the button and clicks it with the mouse, or other device, to initiate that action. In practice, however, these buttons can be fairly complex. Some buttons activate when a user's finger presses down on the mouse. Other buttons merely highlight when this happens, and activate only when the user's finger is released, allowing the user to back out of the action halfway through if desired. More complex buttons may allow multiple actions, depending on pressed keys on the keyboard and the use of different mouse buttons, and some even pop up menus of choices if the user's finger is held down past a certain threshold.

Some of this complexity is transferred to the world of stylus-based systems. All of these types of buttons work well with a mouse. How are they with a stylus? We came to the following conclusions:

- Unlike buttons in many mouse-based systems, buttons in a stylus-based system should activate when pressed, rather than when released.
- Instead of relying on a cursor, like a mouse-based system, stylus buttons should highlight when the stylus tip is positioned above them, to provide feedback to the user.
- Although stylus buttons should unhighlight if the stylus tip is moved away without pressing down, they should activate if the stylus is pressed down near the button when the button is highlighted. That is, the active region of the button should be larger than the image of the button.

Activation when pressed

Although most virtual buttons in mouse based systems activate when released, most physical buttons activate when they are pressed (e.g. a doorbell, a touchtone phone, or a keyboard). Since stylus-based systems are very closely linked to the user, they need to function this way too. We tried using a stylus with buttons that highlight when pressed and activate when released and found them to be very

awkward. Buttons that activate when pressed worked much better.

Highlighting buttons

It turns out that although people sometimes dial a telephone with the eraser of a pencil or the back of a pen, they are unfamiliar with pressing buttons using the tip. The designers of a stylus-based system should not require that users change their grip on the stylus whenever they wish to press a button, so the buttons must be pressed while the stylus is held in a writing position. This is an unfamiliar action, however, and can make it difficult to position the tip of the stylus accurately in a button.

Highlighting the buttons helps. Instead of highlighting a button when it is pressed, as in some mouse-based systems, however, it is much more effective to highlight the button when the stylus is positioned above it. Then the user can have some confidence about what will happen when the stylus tip is pressed down, but can still have the option of backing out of the operation. This requires that the mechanism used to track the stylus can do so even when the stylus tip is not touching the screen surface. Not all stylus tracking mechanisms can do this, but it is important for this sort of application.

The alternative is to have a cursor tracking the position of the stylus tip when the tip is not touching the screen. Positioning the cursor in a button is equivalent to the button highlighting when the pen is above it. We believe, however, as do others, like the makers of the Penpoint system, that a cursor is out of place in a stylus-based system, because it is so natural to indicate a position on the screen with a stylus. Highlighting makes more sense.

Buttons with a large active region

When we built buttons this way, though, we encountered the following problem: When a user is holding the stylus in a writing grip and has positioned the tip of the

stylus over a button and then presses the tip down, the tip sometimes leaves the button before contacting the screen. This is because the action of pressing down with the tip is one of rotating the hand slightly, and can move the tip laterally. This was one of the principle reasons users had trouble pressing buttons that didn't highlight, and they continued to have some trouble, even with buttons that did highlight.

The obvious solution is simply to make the buttons larger, under the assumption that if the buttons were big enough, users would have no problems pressing them. Although this is true, of course, screen real estate on some devices is limited, and having very large buttons may not be desirable. We would like to increase the size of the buttons as little as possible. A solution that works fairly well is to make the region associated with the button larger, but to leave the image of the button the same size. Thus when a user has positioned the tip of the stylus over the image of a button, the image highlights. Now the user can press the tip of the stylus down, and the button will activate as long as the tip is inside the active region of the button. Even if the stylus tip does not contact the screen inside the image of the button, the button will still be pressed because the button's active region is larger than its image. The active region cannot be too large, however, because there must be some way to back out of pressing a button.

To fit more buttons in a restricted area, the active regions can overlap, as long as they do not overlap with the images, because any highlighted button should unhighlight if the user move the stylus tip over the image of another button, causing the new button to highlight. Only one button should be highlighted at a time. An example of this configuration is shown in figure 4-7.

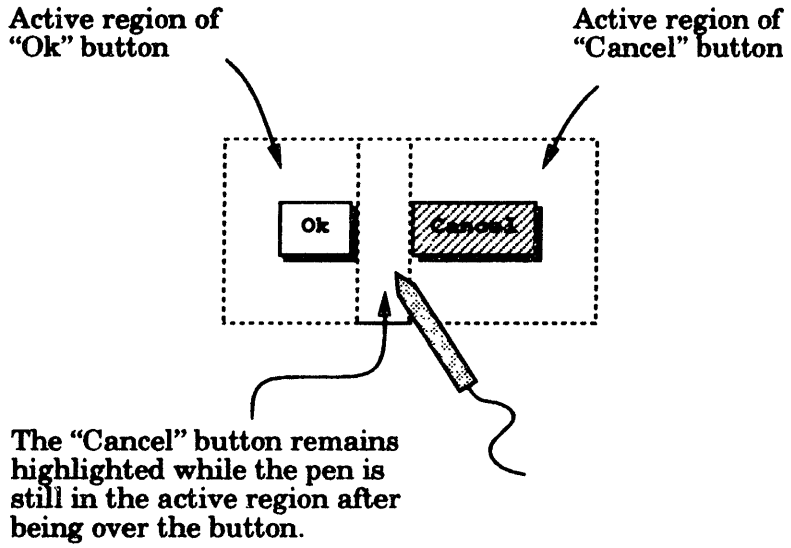

Figure 4-7: Two buttons with larger active regions than their images. The active regions can overlap with each other, but not with the images of the buttons because a button is highlighted once the stylus is positioned over its image, but it does not unhighlight until the stylus tip leaves its active region. This allows users to "miss" when pressing a button and still have the button activate, as long as the button was highlighted when the user pressed the stylus down.

4.8 Summary

Stylus-based editing can be easy and effective. Executing common editing commands by drawing marks on the text to be edited works well. There should not be too many of these marks and they should be easy to make and remember, perhaps simplified proofreaders' marks.

Our system allows the editing of text with a few simple gestures: delete character, delete word(s), insert, and transpose. These gestures are recognized by statistical analysis of the path traced by the stylus tip. For less common edits, a user can select a region of text either by circling it or by marking its ends. The region can then be operated upon.

New text can be entered either in a special mode that is separate from editing, or text entry can be integrated with the editing process. Our system uses modal text entry, but either is feasible.

We asked several users to perform a series of edits on a sample document, and they were able to do so quickly and easily, making use of the various features of the system, including text entry. There is some question, however, about the tasks for which this sort of editing system is best suited.

4.8.1 What's it good for?

What is text editing? To some it is making minor changes to a document that is almost in final form, or changing a document slightly to produce a similar document. To others it encompasses the process of document creation and revision, including large amounts of text entry and major changes. The text editing described in this chapter falls somewhere between these two extremes.

Certainly this type of system is useful for making modifications to an existing document, either for purposes of revision, or in changing a copy into a new document. Deleting and inserting small amounts of text are easy and reorganizing

the information in the document can be done quickly and efficiently. This system would also be good for combining two documents together.

For creating an entirely new document, however, this system may not be ideal. It would be useful for creating documents based on templates or for creating documents that involve a significant amount of material drawn from other sources. But it might not be the best choice for writing a free-form letter or other document with a large amount of new text. Since its primary weakness is in text entry, it will become more useful as it is integrated with more sophisticated text entry techniques, such as voice entry or the features described as future improvements in section 3.6.

Chapter 5

Markup Editing

In this thesis "markup editing" refers to the process of editing a document by making marks on it rather than changing the document directly. This is similar to the process many people go through when editing a document on paper. They read through the document and mark the suggested changes or problem areas, and then produce a new version of the document later.

This practice has some advantages over editing a document on a computer, but has some drawbacks as well. We experimented with implementing this sort of editing system on a stylus-based computer, marking up an electronic version of a document rather than a paper one. This chapter describes our system and discusses some benefits and drawbacks of markup editing.

This is not a topic that has been extensively explored before because most existing editors have been keyboard based. One can imagine a system marking a word as deleted when the key sequence to delete it is pressed (rather than actually removing the word), but there are few, if any, keyboard-based systems that do this. This is probably because no one could think of any advantages such a system would have over one in which edits happen immediately as they are keyed in.

In a stylus-based system, using gestural editing techniques, markup editing makes a lot more sense. When editing on paper, a user might mark a word to

be deleted by crossing it out. That same gesture can be used to delete a word on the screen of a stylus-based editor. Why not combine the two? Instead of actually removing the deleted word, why not simply mark it as deleted?

This makes more sense in a stylus-based system than in a keyboard-based system because the gestures indicating the edits are similar to the marks a user makes to specify those edits. In a keyboard-based system, there would be no relationship between a mark indicating a change and the keystroke that caused that mark to appear.

5.1 A markup editing system

We built such a system, combining the gestural editing techniques described in chapter 4 and the text entry techniques described in chapter 3. Instead of actually changing a document, however, it merely places edit marks over the image of the document.

5.1.1 Marking changes

One way for the system to indicate where changes are to occur is to leave the edit marks drawn by the user on the document. That is, a user might cross out a word, and then instead of the "crossing out" mark disappearing along with the word, as it would in a conventional editor, both would stay. If the user scrolled that part of the document away and returned to it later, the mark would still be there, signifying that the word was to be deleted.

The problem here is that a user cannot tell how the system has interpreted a gesture. As described in chapter 4, gestures can have complex interpretations, and a user may need to know exactly how the system has interpreted a particular gesture. Consequently, our system removes the user's mark and draws a stylized version of the mark to indicate its precise interpretation. This is shown in fig-

ure 5-1. This process is analogous to removing the characters a user draws and redisplaying them in a nice font, as described in chapter 3, so the user knows how the system has interpreted the gestures drawn on the screen.

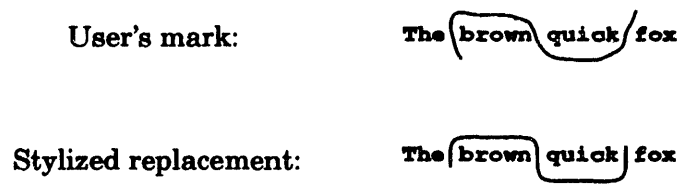

Figure 5-1: A user's mark, in this case indicating a transposition, is replaced by a stylized version of that mark showing precisely what change is to occur.

One concern about this practice is that it may be hard to see the stylized marks, although an advantage of marking up the document rather than changing it is that these changes should be easy to see (this is described further in section 5.2.3 below). If the stylized version of these marks is too much like the underlying text, it may be hard to notice changes. This is an important concern, but one that is easy handled. Because the marks are generated by the system, they can be made arbitrarily different from the underlying text, including (in future systems) being in different colors. Thus the changes can be in high contrast to the text.

Since our system does not have color, we use marks that employ much thicker lines than the text being edited. Figure 5-2 shows some of the marks our system used to signify edits.

5.1.2 Insertions

One of the biggest problems with editing a document on paper is that there is not much room to write new text. If the document is double spaced, there is room for one line of text, and there is sometimes space in the margins, but inserted text is inevitably squashed awkwardly onto the page. An electronic version of this editing

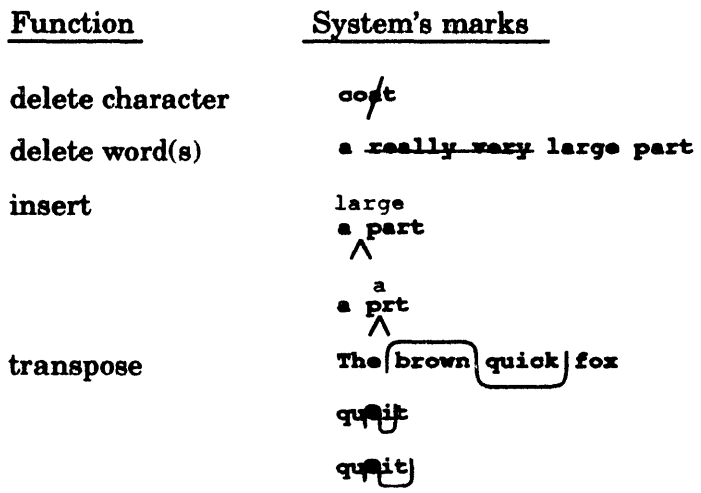

Figure 5-2: Some of the system's editing marks.

practice need not have these problems. In our system, when text is inserted, the insertion point is clearly marked with a stylized caret (as shown in figure 5-2) and the inserted text is positioned between two lines of text, expanding this space if necessary. Figure 5-3 shows a phrase inserted between two lines of text.

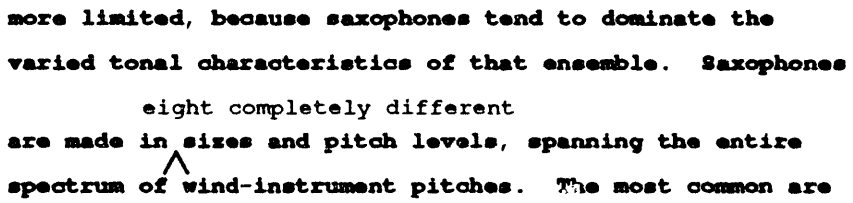

Figure 5-3: Inserted text need not be cramped. Existing lines of text can be pushed apart to make space.

5.1.3 The new document

Because the system knows what change each mark on the document signifies, it can implement those changes at any time to produce a new document. Of course,

the user can still go back and view the marked up document if desired.

5.2 Benefits

Markup editing has many advantages over traditional editing. Essentially, it provides most of the advantages of editing on paper, but without the major drawback that the edits must later be entered into a computer. A markup system provides a record of all changes that have been made to a document. It also allows much more flexible "undoing" than other types of editing. Furthermore, this visible record is a good way to communicate changes to a document or proposed changes that can then be automatically accepted either all at once or selectively.

5.2.1 Audit trail

One major advantage of markup editing is that it leaves a clear audit trail of all of the changes that have been made to the document. Many document revision control systems keep track of periodic versions of a document, but there is generally no record of how one version was changed into another. If many changes were made, it can be quite difficult to follow this trail. With a markup editing system, however, each previous version can also include all the changes that were made to obtain the next version.

In addition, markup editing allows the tracking of changes made by several people, or several people simultaneously. If changes made by different people are displayed in different styles (or colors), it can be easy to look at a document and tell immediately which changes were made by whom. This might be especially useful for collaborative editing systems.

Also, because the changes are not actually made to the document, but only marked on it, a document could be passed from one person to another, with changes being added or removed at each step, and allow a better process of working together.

5.2.2 Non-chronological "undoing"

One feature of many editors is the ability to "undo" some change, either because it was a change made in error (e.g. the user pressed the wrong key) or because the change is no longer desired. The problem with these systems is that it is difficult to describe which change is to be undone. In some systems, only the last change may be undone, and all other changes are permanent. In most systems, though, a record of the past few changes is kept, and these may be undone in reverse chronological order. This is enforced for two reasons. First, changes to a document are often affected by those changes which occurred after them. That is, an earlier change cannot be undone without undoing what came later. Second, a user has no way to specify which edit is to be undone. A system could also present a list of the edits (described in some language) and allow a user to choose one to undo, but this would not solve the problem of overlapping edits and it would require the user to understand the description of each edit.[1]

Because of the enforcement of reverse chronological undoing, most users undo only edits that were made in error, since they notice them almost immediately. If an edit is made and the user realizes immediately that it is not a desirable edit, then it is undone. This is unfortunate because revising a document is an iterative process and a user may go over a document many times making revisions. If some revision is made and then decided against much later, it is often difficult to undo it. Sometimes a user might need to search through a previous version of the document to find an unchanged copy of the section being edited.

Markup editing solves this problem. Because all edits are visible on the document, it is easy to see the original version and the changes, and to specify which change to undo. The user can simply point to the offending edit.

To implement this, we added a gesture that does not produce an editing mark.

[1]The Tioga editor, as described in *A Structural View of the Cedar Programming Environment* ([33]), has this facility.

The "scribble out" gesture, pictured in figure 5-4, simply lifts an edit mark off the page, as if it had been erased. There is no added complexity and none of the edits made later need to be redone. Any changes that are no longer desired can easily be removed, regardless of the order in which they were made.

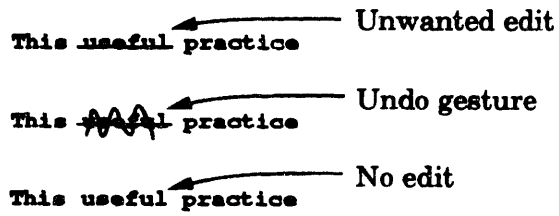

Figure 5-4: The undo gesture removes unwanted edits.

5.2.3 Obvious editing changes

When presented with a new version of a document, people familiar with the old version often wish to know what has changed. The changed sections may be indicated by drawing a line in the margin, but this is not nearly so descriptive as seeing the old version with all the changes marked. Of course, this is not desirable for all purposes, but simply glancing at the marked up version can communicate a great deal about what was changed, and how it was changed.

5.2.4 Automatic acceptance of proposed changes

Finally, this sort of system can be used to review changes proposed by someone else, perhaps a collaborative author. Because the changes have not yet been made, some of them can be rejected (perhaps using the undo gesture described in section 5.2.2), and others can be accepted. But once a set of changes is agreed upon, they are already in the computer and a simple button press effects them.

In general, a markup system has almost all of the advantages of editing on

paper, but with the additional benefit that changes do not need to be entered into a computer.

5.3 Problems with markup editing

Markup editing does have some problems, however. Because of its natural feel, users tend to have higher expectations of such a system than are feasible. And once the changes described by the marks are made, users feel lost without the marks.

5.3.1 Complexity

One of the disadvantages of editing on paper is that it is often difficult to determine what the final version of a document will look or sound like. Because the flow of the text is important, it is sometimes necessary to read through the new version of a sentence or paragraph to see if any further changes need to be made.

On paper this is difficult, but users do it anyway because it is necessary. With a computer, though, this should be easy. Once a change is marked, why not look at the new version to see how it looks or sounds? Of course, this is easy to implement too, but it produces some significant complexity.

If a user makes changes to version n of a document, it may be desirable to look at version $n + 1$ to see those changes and modify or add to them if necessary. In addition, however, users naturally assume that they can make the necessary changes to version $n + 1$ and then go back to version n and view both sets of changes. But how should the second set of changes be displayed? The changes made to version $n + 1$ of the document may not make any sense in version n of the document. Figure 5-5 shows a simple example of how overlapping edits can be difficult to display simultaneously.

This difficulty is not intractable. It is certainly possible to come up with a

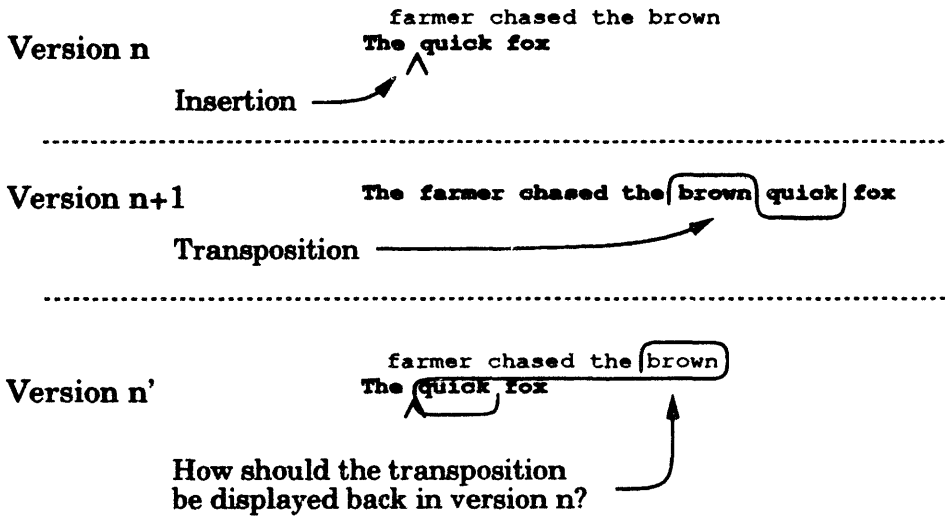

Figure 5-5: Making changes to a changed document. How should overlapping edits be displayed if they were made to different versions of a document? In this example, version n of the document contains an insertion. In version $n + 1$, the user makes a transposition. Returning to version n, how should the transposition be displayed? What is displayed in this figure does not correspond well to the mark the user made to indicate the transposition.

representation of all changes to a document, even if some of those changes were made on a later version of the document, as in figure 5-5. The problem, however, is that the representation of the secondary changes is unlikely to correspond closely to the marks drawn by the user to indicate those changes. It would seem strange to indicate one set of changes and view another. This problem occurs because users naturally expect more from this sort of system than is feasible.

Another complexity introduced by this sort of system is that of major changes to a document. Local changes such as deletions and insertions are easy to show with marks on the page. More dramatic changes, however, are not so easy. How should the document display the fact that chapters two and three have been switched? How should this be shown on a printed version of the document? And what about moving a paragraph from page 27 to page 118? This is very different from moving a paragraph from immediately before a figure to immediately after the figure. Like the problems with overlapping edits, all these difficulties are surmountable, but they introduce complexity into the system that may make it hard for users to understand.

5.3.2 Miraculousness

Once a user is satisfied with the marked changes to a document, all that remains is to push a button to create a new version of the document with all those changes made. Surprisingly, people did not react well to this. When the button is pushed, the document is instantly transformed, miraculously. The system has made all the changes.

The natural reaction was to look for those changes, to confirm that they were actually made and that they had the desired effect. But the changes are very hard to find. There are no marks by which to locate them and the structure of the document may have changed radically. It really is a new document. This is very disconcerting, and several of our test users commented on it. We could

tell when observing them that the change from a version of the document with edits to another version of the document without was shocking. They skimmed through the document, looking for the changes, making sure that the changes had happened correctly.

It is possible that this phenomenon is caused by a lack of confidence in the system, and that as users gain experience with it, they will become more confident and be more comfortable with the idea. The only difference between this and handing a marked up copy of a document to a secretary is the suddenness of it. Nonetheless, stylus-based systems are intended to be easier and more natural to use than existing systems. They should not be designed such that users need to get used to them and significantly change their expectations and the way they think about document processing.

One possible solution to this problem is to use a two-view approach to document editing. This sort of approach has been used before, for example, in a system developed by Kenneth Brooks ([2]), but it has not been applied to markup editing. In a two-view approach to markup editing, there would be two versions of a document on the screen. One would be a version of the document showing the marks, and the other would be a version showing the changes. Thus a user could see the results of the edits, solving some of the problems of complexity described in section 5.3.1, and would also get used to the new version of the document as it was created. There would be no miraculous button press, the user would simply move to the new version.

Of course, this would also have problems. For one thing, in Brooks' system, both versions of the document can be changed and changes in one are reflected in the other. In this system, the user would probably be prohibited from making changes in the new version of the document, to eliminate many of the problems described in section 5.3.1, and this might be frustrating. In addition, users might have the problem of looking back and forth between the two versions, similar to

the problem of moving a hand back and forth between the keyboard and mouse.

5.4 Summary

Although it offers some attractions, markup editing requires a significant amount of study before it will be practical. And even if most of the implementation problems were solved, it still has inherent obstacles to being as useful as might be desired. Its benefits include a clear record of changes to a document, non-chronological "undoing," a good way to present those changes, and an easy way to communicate proposed changes. Its difficulties, however, are significant complexity and the miraculousness of the transformation from a marked up document to a new version of the document.

Chapter 6

Conclusions

Interaction with a stylus-based computer is easy and intuitive. Although entering text is fairly slow and tedious, it is feasible for small amounts of text entry and we found some ways to make it easier. Editing text with a stylus, however, makes a lot of sense and is well suited to "intelligent" techniques in text editing that do not make as much sense in a keyboard-based system. Markup editing has the potential to be useful for some applications, but it is not practical for general text editing, at least not without significant further study.

6.1 Text entry

The main difficulty with text entry by stylus is speed, making it most feasible for such tasks as entering a search string or file name, or adding a word, sentence or paragraph to a document. Without clever methods of reducing the amount of text actually entered by a user, however, it is probably not suitable for the entry of a large amount of text. It needs to include good methods of error prevention and correction that incorporate intelligence on the part of the system, making use of knowledge of the expected input. Allowing users to tap on errors to correct them is effective, and correcting errors by substituting a most likely second choice for an

erroneous character works well, although it is somewhat limited.

In addition, text entry is significantly more useful if it includes methods for editing the text as it is entered. For this reason, as well as to provide good feedback to the user, we favor interpreting the text as it is entered, not when the user has finished entering it. To facilitate this, a user interface technique can be used to segment a user's marks with a stylus into separate characters. Instead of asking users to write into rows of boxes, the "segmenting cursor" is the set of only those boxes that are necessary at any moment, moving with the stylus as text is entered. We also found that a handwriting recognition system can be tuned by the user, during the regular process of entering text, and that this practice improves recognition rates significantly and quickly.

We postulate that text entry with a stylus could be a feasible way to enter large amounts of text if the text entry made use of methods to reduce the amount of text written by a user. (Two such methods are prediction and completion.) A system free from the constraints of a keyboard and mouse, however, might make better use of a combination of a stylus and a voice entry system, since the stylus could be used for correction and editing, and voice for entry of large amounts of text.

6.2 Editing

Text editing with a stylus is natural and easy. It can be designed to take advantage of skills and gestures with which users are already familiar, and as a result the gestures can be simple to learn and remember. Stylus-based text editing also fits well with users' mental models of the editing process, such that single editing gestures often correspond directly to desired changes.

Much of the power of stylus-based editing comes from the use of gestures that include both the specification of an instruction and the parameters to that instruction. Simplified proofreaders' marks work well as the basis for these gestures. In

addition, various methods can be used to select regions of text: Circling is good for small regions, and marking the beginning and end of a large region also works well. All of these gestures can be recognized using statistical analysis of the path traced by the tip of the stylus and interpreted using heuristics incorporating some knowledge of the editing domain.

If designed appropriately, these heuristics can incorporate a tremendous amount of intelligence to greatly simplify the editing task. Although we had the opportunity to implement only a sample of such intelligence, we are confident that computer systems can be given more sophisticated understanding of the material being edited and take on more of the editing process, requiring users to specify less of the specifics of changes.

In general, we consider a stylus-based system to be better suited to the editing task than a system with a keyboard and mouse. The principle drawback of a stylus-based system is its text entry facility, whether integrated into the system or presented as a separate mode. A stylus will become an even better input device if the problem of fast text entry can be resolved, either using sophisticated methods of text entry with a stylus, or incorporating other technologies, such as voice entry.

6.3 Markup editing

Although markup editing is a satisfying way to modify a document, it presents some significant problems that stand in the way of its being used as a general editing technique. Because it is difficult for users to understand the inherent constraints of such a system, they often expect more than is feasible or can be presented simply. They also find the transition from a marked up document to a new document in which the specified changes have been made to be a sudden and confusing one.

On the other hand, markup editing provides a clear record of all changes made

to a document and allows easy "undoing" of changes in any order. It might be well suited for use as part of an annotation system for proposing changes to a document or as a sophisticated display of the differences between two documents. In such a system, replacing a user's annotations with stylized versions of those marks provides the same benefits as interpreting hand drawn text: it is immediately clear to the user how the system has interpreted each mark and the computer generated marks are more compact and easier to manipulate than the user's originals.

Markup editing is an interesting technique, but requires significant study. As it stands, it seems to be poorly suited for general editing.

6.4 Other interaction techniques

In the process of exploring text entry and text editing with a stylus, we found that standard user interface techniques require adjustment to be effective in a stylus-based system. Some of the interactions that work well with a mouse do not work as well with a stylus, and a stylus allows some interactions that don't make as much sense with a mouse. In general, we assume that most interaction techniques require some adjustment to be adapted for a stylus-based system, but we did not find that, in general, they were more or less effective in a stylus-based system than in a mouse-based system.

The two examples we explored were scrolling and on-screen buttons. Scrolling in a stylus-based system is more effective if the focus of the scroll bar is on moving the document while the screen remains stationary, rather than moving a virtual screen over a stationary document, as is done in most scrolling systems. This is because the text on the screen should move the same direction as the stylus tip when scrolling, to take into account the proximity to the screen of a user's hand and focus.

On-screen buttons in a stylus-based system should highlight when the stylus is

positioned above them and activate when the stylus is pressed down, unlike some mouse-based buttons. In addition, the region in which the stylus tip can be put down to cause the button's function to occur should be larger than the image of the button, so that a user can "miss" a button by a little, but still get the function. This is to compensate for the difficulties of pressing buttons with the tip of a stylus held in a writing position.

6.5 Areas for future study

There are many areas related to this thesis in which future study would be useful and interesting. In text entry, is it possible to eliminate the need for writing in boxes altogether? To what extent should a method for segmenting user's marks into characters be algorithmic as opposed to being a user interface technique. What are good ways for the system to correct errors once a user has pointed them out? How can a text recognition system be tuned automatically, during text entry, but without a user's intervention? And how can the amount of text actually necessary for a user to write be reduced? Is this even a reasonable technique, or does it make more sense simply to use another input device for the entry of large amounts of text?

How should text entry be integrated with text editing? It is acceptable to pop up a text entry window every time text needs to be inserted, but a text editing system might be easier to use if there were no special mode for entering text. Can markup editing be used effectively as a general editing technique, or is it suited only for annotation systems. How should markup editing be used in an annotation system?

Scrolling and on-screen buttons should function differently in a stylus-based system than they do in a mouse-based system. It is likely that other user interface techniques should also. For example, how should stylus-based menus work? Are

pull-down menus appropriate, or should applications rely instead on palettes?

Finally, how can a stylus-based system be adapted and modified to make it more effective? For example, a stylus could have a button on the side. Our experiences were that such buttons were difficult to hold down while drawing or writing, and consequently might be of little use, but this is certainly worthy of further study. Also, a stylus could be integrated with some other user interface device, such as a voice entry system, and used both for editing and for correcting errors in that system (e.g. selecting from among homonyms). Clearly, this also requires significant study.

6.6 Overall

We found stylus-based interaction techniques to be easy to learn, easy to use, and quite effective. We believe that they are applicable to a wide range of systems, varying in size, environment, portability, and other aspects. It is important, as these systems are developed, that they remain simple to use, utilizing only a few easy to make and easy to remember gestures that correspond, both in shape and in traditional meaning, to users' intended actions. Complex and sophisticated processes should be performed primarily by means of intelligence on the part of the system, rather than requiring them to be specified in detail by a user. A stylus-based user interface lends itself to simple instructions by a user resulting in complex behavior by the system. In text editing and entry, at least, we found this to hold up quite well.

Appendix A

Specifics of gesture recognition

Functions computed on user marks

bb_w = **bounding box width**

bb_h = **bounding box height**

$f_1 = |\sum x|$

$f_2 = |\sum y|$

$f_3 = \sum |x|$

$f_4 = \sum |y|$

Tests comparing function values

$t_1 \equiv f_1 < bb_h$

$t_2 \equiv f_4 < \frac{1}{3} bb_w$

$t_3 \equiv f_1 < \frac{1}{3} f_3$

$t_4 \equiv f_2 < \frac{1}{2} f_4$

$t_5 \equiv \mathbf{sgn}(y_{start} - \bar{y}) \neq \mathbf{sgn}(y_{end} - \bar{y})$

$t_6 \equiv f_4 < bb_h \cdot 3$

Values of tests for each gesture

	t_1	t_2	t_3	t_4	t_5	t_6
Delete character	T	-	-	F	-	-
Delete word(s)	F	T	F	-	-	-
Transpose	-	F	F	T	T	T
Insert	-	F	F	T	F	T
Circle (select)	-	-	T	T	-	-
Scribble (undo)	-	F	F	T	-	F
Delineate (select)	T	-	T	F	-	-

Appendix B

Test document for editing

The family of saxophones patented in 1846 by Adolphe Sax combines the single reed of teh clarinet with the bore and fingering patterns of the oboe, producing the tonal qualities of neither. The inrstuments fit into well bands, for their sound blends well with brass and wood wind instruments; their application to the orcestra has been more limited, because saxophones tend to dominate the varied tonal characteristics of that ensemble. Saxophones are made in sizes and pitch levels, spanning the entire specterum of wind-instrument pitches. The most common are the alto and tenor saxophones. They have been effectively used in jazz bands and popular dance orchestras. They also make an excellent vase or small planter and are sometimes used to serve beverages, especially ginger ale or sparkling punch. Numerous jazz perfeormrs have risen to fame with the instrument, and composers, beginning in 19th-century France, have employed it in their solo or ensemble compositions, or as a wall decoration. ELWYN A. WIENANDT

Transpose "into" and "well"

Insert "eight" before sizes and pitch levels.

Remove the sentence about ginger ale.

Remove the phrase about wall decoration.

There are five spelling errors.

Make any other desired changes.

Bibliography

[1] R. Baecker and W. A. S. Buxton. *Readings in Human-Computer Interaction – A Multidisciplinary Approach*. Morgan Kaufmann, Los Altos, CA, 1987.

[2] Kenneth P. Brooks. A two-view document editor with user-definable document structure. Technical Report 33, Digital Equipment Corporation Systems Research Center, Palo Alto, California, November 1988.

[3] Ed Brown, William A. S. Buxton, and Kevin Murtagh. Windows on tablets as a means of achieving virtual input devices. In *Proceedings of Interact '90*, Cambridge, England, August 1990.

[4] D. J. Burr. Designing a handwriting reader. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(5):554–559, September 1983.

[5] William A. S. Buxton, Eugene Fiume, Ralph Hill, Alison Lee, and Carson Woo. Continuous hand-gesture driven input. In *Proceedings of Graphics Interface '83*, pages 191–195, 1983.

[6] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, July 1980.

[7] Luca Cardelli. Building user interfaces by direct manipulation. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages

152–166, Banff, Alberta, Canada, October 1988. Association for Computing Machinery.

[8] Robert Carr and Dan Shafer. *The power of Penpoint*. Addison-Wesley, 1991.

[9] Robert M. Carr. The point of the pen. *Byte*, 16(2):211–221, February 1991.

[10] Pehong Chen and Michael A. Harrison. Multiple representation document development. *Computer*, 21(1):15–31, January 1988.

[11] Michael L. Coleman. Text editing on a graphic display device using hand-drawn proofreader's symbols. In M. Faiman and J. Nievergelt, editors, *Pertinent Concepts in Computer Graphics, Proceedings of the 2nd University of Illinois Conference on Computer Graphics*, pages 282–290, Urbana, Illinois, 1969. University of Illinois Press.

[12] John J. Darragh, Ian H. Witten, and Mark L. James. The reactive keyboard: A predictive typing aid. *IEEE Computer*, pages 41–49, November 1990.

[13] Wolfgang Doster and Richard Oed. Word processing with on-line script recognition. *IEEE Micro*, 4(5):36–43, October 1984.

[14] Roger W. Ehrich and Kenneth J. Koehler. Experiments in the contextual recognition of cursive script. *IEEE Transactions on Computers*, c-24(1):182–194, January 1975.

[15] James D. Foley, Victor L. Wallace, and Peggy Chan. The human factors of computer graphics interaction techniques. *IEEE Computer Graphics and Applications*, 4(11):13–48, November 1984.

[16] Nancy C. Goodwin. Cursor positioning on an electronic display using lightpen, lightgun, or keyboard for three basic tasks. *Human Factors*, 17(3):289–295, 1975.

[17] John D. Gould and Lizette Alfaro. Revising documents with text editors, handwriting-recognition, and speech-recognition systems. *Human Factors*, 26(4):391–406, August 1984.

[18] John D. Gould and Josiane Salaun. Behavioral experiments on handmarkings. *ACM Transactions on Office Information Systems*, 5(4):358–377, October 1987.

[19] Jeffrey C. Jackson and Renate J. Roske-Hofstrand. Circling: A method of mouse-based selection without button presses. In *CHI '89 Proceedings*, pages 161–166. Association of Computing Machinery, 1989.

[20] Arto Kankaanpaa. FIDS – A flat panel interactive display system. *IEEE Computer Graphics and Applications*, pages 71–82, March 1988.

[21] John Karat, James E. McDonald, and Matt Anderson. A comparison of menu selection techniques: Touch panel, mouse, and keyboard. *International Journal of Man-Machine Studies*, 25(1):73–88, 1986.

[22] Lloyd K. Konneker. A graphical interaction technique which uses gestures. In Dr. Robert W. Taylor, editor, *Proceedings, IEEE First International Conference on Office Automation*, pages 51–55, December 1984.

[23] Gordon Kurtenbach and Bill Buxton. GEdit: A test bed for editing by contiguous gestures. Technical report, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, 1990.

[24] H. Kučera and W. N. Francis. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI, 1967.

[25] Palmer Morrel-Samuels. Clarifying the distinction between lexical and gestural commands. *International Journal of Man-Machine Studies*, 32(5):581–590, 1990.

[26] James R. Rhyne. Dialogue management for gestural interfaces. *Computer Graphics*, 21(2):137–142, April 1987.

[27] Dean Harris Rubine. *The Automatic Recognition of Gestures*. PhD thesis, Carnegie Mellon University, November 1990.

[28] Abigail J. Sellen, Gordon P. Kurtenbach, and William A. S. Buxton. The role of visual and kinesthetic feedback in the prevention of mode errors. In *Proceedings of Interact '90*, Cambridge, England, August 1990.

[29] Ben Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, August 1983.

[30] Mark Stefik. Toward the digital workspace. Unpublished manuscript, December 1990.

[31] Ching Y. Suen, Marc Berthod, and Shunji Mori. Automatic recognition of hand-printed characters – the state of the art. *Proceedings of the IEEE*, 68(4):469–487, April 1980.

[32] Ivan E. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proceedings – Spring Joint Computer Conference*, pages 329–345, 1963.

[33] Danial C. Swinehart, Polle T. Zellweger, Richard J. Beach, and Robert B. Hagmann. A structural view of the cedar programming environment. In *ACM Transactions on Programming Languages and Systems*, October 1986.

[34] C. C. Tappert, C. Y. Suen, and T. Wakahara. On-line handwriting recognition – a survey. In *The 9th International Conference on Pattern Recognition*, Rome, Italy, November 1988. IEEE.

[35] Jean Renard Ward and Barry Blesser. Interactive recognition of handprinted characters for computer input. *IEEE Computer Graphics and Applications*, pages 24–37, September 1985.

[36] L. K. Welbourn and R. J. Whitrow. A gesture based text editor. In *Proceedings of the British Computer Society*, pages 363–371. Cambridge University Press, 1988.

[37] D. Whitfield, R. G. Ball, and J. M. Bird. Some comparisons of on-display and off-display touch input devices for interaction with computer generated displays. *Ergonomics*, 26(11):1033–1054, 1983.

[38] Catherine G. Wolf. Can people use gesture commands? Technical Report Research Report RC11867, IBM's Thomas J. Watson Research Center, Yorktown Heights, New York, April 1986.

[39] Catherine G. Wolf and Palmer Morrel-Samuels. The use of hand-drawn gestures for text-editing. *International Journal of Man-Machine Studies*, 27(1):91–102, July 1987.